Stručni rad
https://doi.org/10.31784/zvr.11.1.16
Datum primitka rada: 28. 6. 2022.
Datum prihvaćanja rada: 2. 12. 2022.

OPTIMIZACIJA SQL UPITA NA PRIMJERU AWS SERVISA

Jasmina Diković
Studentica, Veleučilište u Rijeci, Vukovarska 58, 51 000 Rijeka, Hrvatska;
e-mail: jdikovic@veleri.hr

Ida Panev
Dr. sc., viša predavačica, Veleučilište u Rijeci, Vukovarska 58, 51 000 Rijeka, Hrvatska;
e-mail: ipanev@veleri.hr

SAŽETAK

Količina nastalih informacija svakim se danom povećava, a time rastu i zahtjevi za kapacitetom računalnih resursa kojima se one obrađuju i analiziraju. Rastuću ulogu u obradi podataka ima računarstvo u oblaku. Budući da se korištenje usluga u oblaku uobičajeno temelji na plaćanju po potrošnji, poznavanjem mogućnosti optimizacije i efikasnog korištenja resursa moguće je upravljati željenim performansama i troškovima. U ovom radu opisani su i prikazani na praktičnom primjeru različiti postupci koji se mogu provesti u svrhu optimizacije podataka i SQL upita, a prilikom korištenja usluga u oblaku za rad s velikim podacima. Opisani postupci uključuju odabir formata i provedbu kompresije podataka, ali i razne druge mogućnosti značajne za rad u oblaku, poput particioniranja podataka i planiranja strukture SQL upita. U praktičnom dijelu rada, nad podacima različitih obilježja proveden je niz SQL upita kako bi se usporedile performanse u odnosu na primijenjene postupke i obilježja upita. Temeljem navedenih postupaka i prikupljenih rezultata, može se zaključiti kako postoji veći broj mogućnosti putem kojih se može optimizirati provedba SQL upita. Važnost ovih postupaka sastoji se u njihovom doprinosu učinkovitom i isplativom korištenju usluga u oblaku prilikom rada s velikim podacima.

Ključne riječi: *SQL, optimizacija, veliki podaci, računarstvo u oblaku, Amazon Web Services*

1. UVOD

U suvremenom poslovanju sve veći udio uspjeha organizacija ovisi o kvaliteti informacija na temelju kojih se donose odluke. Količina novih i obrađenih informacija svakim danom sve se više povećava, što znači da se postavljaju i sve veći zahtjevi po pitanju kapaciteta resursa kojima se te informacije analiziraju. Važnu ulogu i potencijal u upravljanju podacima ima računarstvo u oblaku (engl. *cloud computing*), budući da korisnicima pruža resurse i mogućnosti dostupne bez

posjedovanja infrastrukture za obradu velike količine podataka. Kako je riječ o modelu poslovanja koji se uobičajeno temelji na plaćanju po potrošnji, u korištenju ovih usluga vrlo je bitno poznavati načine kako efikasno koristiti resurse, s obzirom na to da postoji niz postupaka kojima se mogu poboljšati performanse i smanjiti troškovi korištenja. Analiza podataka u bazama podataka često se provodi koristeći SQL (engl. *Structured Query Language*), koji uključuje širok raspon mogućnosti i postupaka te značajan prostor za optimizaciju u prethodno spomenute svrhe: bolje performanse postupaka, smanjenje troškova i kraće vrijeme potrebno za provođenje analitičkih aktivnosti. Optimizacija podataka i SQL upita tako predstavlja moćno sredstvo u efikasnom i isplativom korištenju usluga u oblaku za analizu velikih količina podataka. U prvom dijelu rada prikazana su najbitnija obilježja velikih podataka (engl. *big data*), pojam računarstva u oblaku i određeni servisi koje pruža AWS (engl. *Amazon Web Services*), SQL obilježja važna za analitiku velikih podataka te postupci kojima se provođenje upita može poboljšati. Drugi dio rada uključuje pripremu jednog seta podataka za analizu u odabranim AWS servisima, a nad kojim su provedeni različiti SQL upiti kako bi se usporedile performanse u odnosu na primijenjene postupke i obilježja upita.

2. PREGLED LITERATURE

Razvoj u području upravljanja podacima potaknuo je znatan broj istraživanja usmjerenih na ispitivanje specifičnosti i učinkovitosti različitih sustava baza podataka. Razvojem metoda istraživanja pohrane i upravljanja podacima sve veći značaj dobivaju ključne metrike njihove učinkovitosti, među kojima su neke od najzastupljenijih vrijeme izvođenja upita, procesorsko opterećenje ili memorijski zahtjevi (Le i sur., 2015). Uslijed rasta računarstva u oblaku koje obuhvaća resurse na zahtjev, dodatnu važnost imaju troškovni pokazatelji, posebice zbog količine podataka koje takvi sustavi mogu obrađivati. Područje optimizacije rada pri korištenju računarstva u oblaku sve je opsežnije, što primjerice potvrđuje i činjenica kako Google Znalac (engl. *Google Scholar*) pretraživanjem izraza „*cloud computing optimization*" (optimizacija računarstva u oblaku) samo od 2021. godine do danas već može pronaći više od 50 000 rezultata. Optimizacija rada u oblaku dotiče se različitih područja, uključujući modele podataka i infrastrukturu resursa, primjenu suvremenih tehnologija poput strojnog učenja i Interneta stvari, energetsku učinkovitost i stupanj iskorištenosti resursa, a udio istraživačkog rada usmjeren je i prema optimalnom korištenju resursa u odnosu na područje namjene i poslovne primjene (kao što su primjerice sigurnost, razvoj aplikacija, upravljanje podacima). U jednom od nedavnih pregleda literature u ovom području (Ashari i sur., 2021), autori pronalaze kako su za suvremene sustave u upravljanju podacima važna obilježja velikih podataka (budući da značajke ovih podataka mogu utjecati na učinak i troškove sustava), optimizacija baza podataka, SQL i NoSQL tehnike i postupci te značajke računarstva u oblaku. Zastupljenost spomenutih faktora vidljiva je na primjeru različitih istraživanja. Primjerice, u jednom istraživanju optimizacije upita nad bazama podataka u oblaku (Jain, Raghu i Khanna, 2021), autori zaključuju kako je ova tema višefaktorski problem, a smatraju kako se vremenska složenost može značajno smanjiti primjenom tehnika optimizacije. U nedavnom istraživanju rješenja za optimizaciju kompleksnih upita u oblaku, Sellami i Defude (2018) pristupaju temi kroz aspekte poput shema podataka, modela evaluacije troškova i optimiziranja plana izvršenja upita. Sharma i Kaur (2020) su za analizu performansi Hive upita uspoređivali različite tehnike optimizacije podataka, poput particioniranja i *bucket* metode kako bi se unaprijedilo vrijeme provedbe

upita. Za pohranu podataka koristili su AWS servise, a zaključuju kako se primjenom postupka particioniranja može značajno poboljšati vrijeme izvođenja upita. Liu i suradnici (2018) bavili su se usporedbom performansi prilikom primjene tehnika poput korištenja indeksa i stupčastih formata podataka pri testiranju različitih skupova podataka i upita. U svom preglednom radu, Nerić i Sarajlić (2020) bave se prikazom koncepata, tehnologija i tehnika optimizacije pomoću kojih se mogu poboljšati performanse prilikom obrade velikih količina podataka. Neke od tehnika za Hive upite koje opisuju su paralelno izvođenje upita, kompresija, format podataka, particioniranje ili optimizacije temeljene na troškovima. Iz pregleda literature mogao bi se izdvojiti niz koncepata koji se proučavaju u smislu optimizacije u oblaku, iako se u pravilu promatrane performanse dotiču već spomenutih aspekata: obilježja velikih podataka, postupaka na razini sustava za upravljanje bazama podataka i specifičnosti rada sa servisima u oblaku.

3. VELIKI PODACI: POJAM I OBILJEŽJA

Veliki podaci mogu se definirati kao skup opsežnih i kompleksnih podataka, koje zbog svoje veličine ili strukture nije moguće obraditi tradicionalnim alatima i softverima za obradu podataka (Dumbill, 2012). Izvori ovakvih podataka mogu biti različiti sustavi poput društvenih mreža, web poslužitelja, senzora, uređaja, dokumenata i raznih drugih kanala kojima se prikupljaju podaci o cijelom nizu ljudskih aktivnosti i djelovanja. Jedan od izvora koji doprinosi brzom rastu količine registriranih podataka je Internet stvari (engl. *Internet Of Things, IoT*), omogućavajući povezivanje fizičkih događaja i njihovog digitalnog zapisa kroz mrežu senzora različitih uređaja putem kojih se podaci prikupljaju.

Jedna od prvih definicija velikih podataka, koja se naziva i Gartnerova 3V interpretacija, sagledava tri dimenzije koje određuju ove podatke: količinu podataka – *volume*, brzinu nastanka i korištenja podataka – *velocity* i raznolikost strukture ili formata podataka – *variety* (Buyya, Calheiros i Dastjerdi, 2016). Isti autori opisuju kako je s vremenom ova definicija proširivana na još veći broj „V" svojstava, pa sve do 10V ovisno o svojstvima koja se uzimaju u definiciju.

Zbog prednosti koje pruža, sve je veći udio organizacija koje za potrebe velikih podataka počinju koristiti usluge računarstva u oblaku (Statista Research Department, 2022). Kako je za obradu velikih količina podataka potreban i odgovarajući računalni te skladišni kapacitet, korištenje rješenja u oblaku znači veće mogućnosti i za manje organizacije koje vlastitom infrastrukturom ili drugom vrstom kapaciteta inače ne bi mogle podržati analitičke zahtjeve velikih podataka. S obzirom na raznovrsnost usluga u oblaku u pogledu raspona, modela poslovanja, dostupnih resursa i načina plaćanja te zastupljenosti, podatak o rastućem trendu računarstva u oblaku čini se kao logičan ishod njegovog razvoja.

4. RAČUNARSTVO U OBLAKU

Računarstvo u oblaku može se definirati kao model i pristup skupu računalnih resursa u domeni informatičkih djelatnosti koji se isporučuju kao usluga koristeći internetske tehnologije (Gartner Glossary, 2022). Većina modela računarstva u oblaku ne zahtijeva posjedovanje fizičkih resursa poput hardvera i infrastrukture, a usluge su korisnicima dostupne na zahtjev te se uobičajeno

plaćaju prema potrošnji. Računarstvo u oblaku rastući je trend u poslovanju organizacija i uporabi kod privatnih korisnika, a prednosti uključuju i smanjene troškove održavanja sustava, fleksibilniji način rada, dostupnost podatkovnih kapaciteta te snažnih računalnih resursa (Sander, 2021). Sve veći udio ovih usluga koristi se za testiranje i razvoj aplikacija, upravljanje velikim setovima podataka, obradu podataka te primjenu u specifičnim područjima poput umjetne inteligencije, strojnog učenja ili napredne analitike (PCSC, 2018).

Usluge računarstva u oblaku mogu se podijeliti prema više kriterija, a među uobičajene spadaju oblici i modeli pružanja usluga. Najčešći modeli su Iaas (engl. *Infrastructure-as-a-Service*), PaaS (engl. *Platform-as-a-Service*) i SaaS (engl. *Software-as-a-Service*). IaaS označava model u kojemu se korisniku isporučuje infrastruktura, uključujući poslužitelje, mrežu, operativne sustave i pohranu putem virtualizacije (Krmpotić, 2020). Model PaaS odnosi se na usluge platforme u oblaku, što korisniku omogućuje korištenje hardvera, softvera i infrastrukture u svrhu razvoja, izvođenja i upravljanja aplikacijama, bez potrebe za izgradnjom i održavanjem takve platforme na lokaciji korisnika (IBM Cloud, 2020). SaaS označava upravljanje svim aplikativnim resursima korisnika, od mreže, spremišta i servera do podataka i same aplikacije (Krmpotić, 2020), a može se opisati kao usluga aplikacije u oblaku.

Na tržištu računarstva u oblaku za poslovne korisnike, izdvajaju se dvije tvrtke koje predstavljaju većinu tržišta: AWS i Microsoft Azure. Iako su prisutni i drugi pružatelji usluga kao što su Google Cloud, Alibaba i IBM Cloud, njihovi su udjeli znatno manji. Zastupljenost AWS-a proizlazi iz činjenice da uključuje širok raspon usluga, poput na primjer analitičkih alata, mrežnih usluga, razvojnih alata, sigurnosti, poslovnih aplikacija i raznih drugih – AWS uključuje više od 200 pojedinačnih usluga u domeni računarstva u oblaku (Amazon Web Services, 2022 d). Microsoft Azure stekao je velik broj korisnika jer nudi kombinacije s uslugama poput MS Teams i Office 365 platforme, koje su popularne za organizacijske potrebe (Vailshery, 2022). Odabir pružatelja usluga računarstva u oblaku ovisi o mnogo faktora, među kojima su neki od najvažnijih pouzdanost pružatelja, financijski aspekt, performanse, sigurnost i prikladnost u odnosu na svrhu korištenja (Rösch, 2016). AWS je trenutno vodeći pružatelj usluga po tržišnom udjelu i opsegu servisa, a ključan razlog za odabir u ovom radu proizlazi iz usporedbi koje pokazuju njegove prednosti za upravljanje velikim podacima (ProjectPro, 2022). Usporedbe pokazuju kako u ovom području AWS u odnosu na Microsoft Azure ima razvijeniju infrastrukturu i zreliji sustav za okruženje velikih podataka te razvijenije mogućnosti za potrebe skladištenja takvih podataka (Data Semantics, 2021). Dodatan razlog za ovaj odabir uključuje i procijenjenu jednostavnost korištenja te visok stupanj dokumentiranosti (BasuMallick, 2022) koji olakšavaju primjenu u svrhu provedbe odabranog primjera. Budući da su za postupke u sklopu ovog rada korišteni AWS servisi, u nastavku su detaljnije razmotrena obilježja ovog pružatelja i pojedinih uporabljenih usluga.

5. AWS – AMAZON WEB SERVICES

AWS označava platformu računarstva u oblaku koju pruža Amazon, a koja uključuje niz različitih usluga iz IaaS, PaaS i SaaS modela (Gillis, 2020). Usluge obuhvaćaju različite kategorije resursa: obradu i skladištenje podataka, mrežne kapacitete, sigurnost, analitiku, umjetnu inteligenciju i

razne druge. Pojedinačni servisi mogu se integrirati na raznolike načine ovisno o ciljevima korisnika, a neki od brojnih servisa su primjerice (Amazon Web Services, 2022 d):

Analitika: Amazon Athena, Data Pipeline, Glue;
Razvoj aplikacija i računarstvo: Amazon EC2, Elastic Beanstalk, Lambda;
Baze podataka: Amazon Aurora, Redshift, DynamoDB;
Skladištenje i pohrana podataka: Amazon S3, Backup, DRS.

Servis Amazon S3 omogućava pohranu objekata koji se mogu koristiti za različite namjene, poput stvaranja podatkovnog jezera, pohrane podataka za aplikacije, sigurnosne kopije podataka ili analitiku velikih podataka (Amazon Web Services, 2022 a). Podatke koji se nalaze u S3 moguće je prema potrebi organizirati, optimizirati i konfigurirati im postavke pristupa ovisno o zahtjevima korisnika. Podaci se skladište na način da se organiziraju u spremnike (engl. *bucket*) kao objekti, pri čemu objekt može biti bilo kakva datoteka (tekstualna datoteka, slika, video, itd.) ili metapodaci o datotekama.

Amazon Athena je servis za postavljanje SQL upita putem kojega se podaci mogu analizirati izravno iz Amazon S3 servisa (Amazon Web Services, 2022 b). Tablice i baze podataka prikazane u Atheni zapravo su spremnici definicija metapodataka koji sačinjavaju shemu izvornih podataka i usmjeravaju Athenu na S3 lokaciju gdje se podaci nalaze. Može se integrirati s AWS Glue katalogom podataka (engl. *AWS Glue Data Catalog*) koji omogućava rad s metapodacima S3 podataka (poput sheme i tipova podataka). Pomoću njih je moguće kreirati tablice i postavljati SQL upite u Atheni, a koji se dodatno mogu koristiti s integriranim mogućnostima AWS Glue servisa za analizu podataka i ETL (engl. *extract-transform-load*)[1].

AWS Glue nudi mogućnost upravljanja ETL procesima, a služi za otkrivanje podataka, transformaciju i pripremu podataka za pretraživanje i postavljanje upita (Amazon Web Services, 2022 c). Zasniva se na interakciji većeg broja komponenti. Jedan od važnijih je ranije spomenuti središnji repozitorij metapodataka, AWS Glue katalog podataka, dok *job* predstavlja poslovnu logiku potrebnu za provođenje ETL akcija. *Crawler* je komponenta koja služi za utvrđivanje sheme podataka i stvaranje tablice metapodataka u AWS Glue katalogu. Nakon što se nad podacima provedu potrebne akcije u AWS Glue servisu, podaci su spremni za korisnika koji ih može dohvatiti i analizirati u drugim servisima, najčešće koristeći SQL upite i njihove mogućnosti.

6. OPTIMIZACIJA PODATAKA I SQL UPITA

SQL je do danas postao ključni jezik za rad s bazama podataka, a predstavlja strukturirani upitni jezik za postavljanje upita nad bazama podataka (Mujadžević, 2016). Primijenjuje se kroz veći broj sustava za upravljanje bazama podataka, kao što su primjerice Microsoft SQL Server, MySQL,

[1] ETL, skraćenica od *extract-transform-load*, u prijevodu označava ekstrakciju, transformaciju i učitavanje podataka. ETL sustav dohvaća podatke iz izvorišnih sustava, primijenjuje standarde za kvalitetu i konzistentnost podataka, prilagođava podatke na način da se različiti izvori podataka mogu zajednički koristiti, a sve s ciljem dostavljanja podataka u formatu namijenjenom uporabi od strane krajnjih korisnika (Vukelić, 2014).

PostgreSQL ili Oracle. Iako postoje standardi za SQL jezik, svaki sustav može imati različite varijante i nadogradnje osnovnog SQL jezika. Uz funkcije i izraze, u pripremi za korištenje SQL-a može se primijeniti cijeli niz postupaka za poboljšanje učinkovitosti. Pojam optimizacije upita ili poboljšanja performansi široka je tema, zato što obuhvaća različita područja putem kojih se postupci optimizacije mogu provoditi; od hardvera, mrežnih postavki, memorije, alata, okruženja pa do samih podataka i primjene postupaka na razini upita. S obzirom na ciljeve ovog rada, naglasak u nastavku je na podacima i upitima.

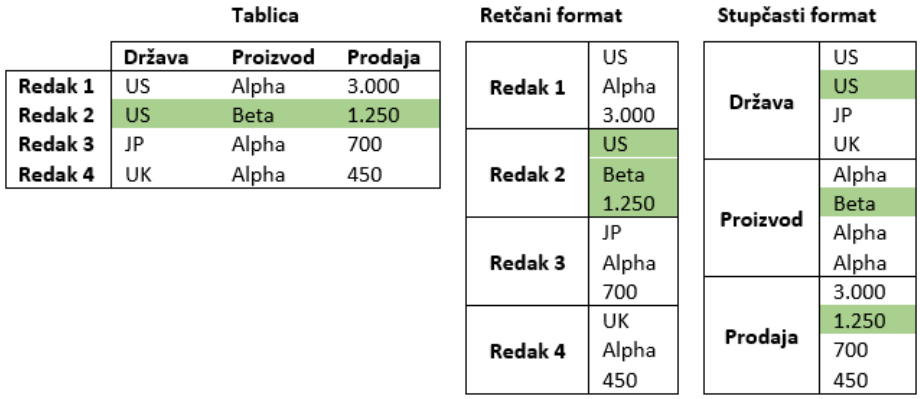
Rad s velikim podacima donosi određene izazove na razini ulaza i izlaza pohranjenih podataka. Osim same količine podataka, prikupljanje podataka iz mnoštva izvora također znači kako podaci ne moraju biti strukturirani, već mogu biti polustrukturirani ili nestrukturirani i pristizati u različitim formatima. Vrijeme potrebno za obradu podataka također je važan faktor, a izravno je povezano s veličinom seta podataka koji se skenira u upitima i načinom na koji su dohvaćeni podaci organizirani u spremištima.

6. 1 Format podataka

Za odabir formata podataka moguće je voditi se različitim ciljevima koji se žele postići i obilježjima radnih zahtjeva. Oni mogu uključivati dostupno okruženje (i poznavanje podržanih formata), predviđanje hoće li se struktura podataka mijenjati s vremenom, očekivanu količinu podataka, broj i vrstu zahtjeva za čitanje/pisanje podataka ili potrebu za izvozom baze u drugo okruženje (Pal, 2016). Jedna od ključnih podjela formata podataka razlikuje pohranu u redcima (engl. *row data storage*) i pohranu u stupcima (engl. *columnar data storage*), a primjer usporedbe prikazan je na slici 1. Vidljivo je kako pohrana u redcima predstavlja „tradicionalan" način skladištenja, u kojemu se podaci pohranjuju slijedno kao niz slogova s podacima redaka tablice (redak 1, 2, 3, 4). Za razliku od toga, u stupčastom formatu pohranjuju se podaci iz stupaca prikazane tablice (u ovom primjeru tri stupca: država, proizvod, prodaja) na susjednim memorijskim lokacijama. U smislu primjene, stupčasti formati primjenjiviji su kod zahtjevnijih analitičkih upita za čitanjem podataka, dok su redčani formati bolji odabir za česte transakcijske upite koji trebaju zapisati podatke (Woodie, 2018).

Najčešći redčani formati za velike podatke su CSV (engl. *Comma-Separated Values*), JSON (engl. *JavaScript Object Notation*) i AVRO. CSV i JSON su tekstualni formati čitljivi čovjeku, ali s druge strane imaju određena ograničenja (npr. značajnije izmjene sheme, mogućnosti kompresije) i općenito mogu biti manje efikasni za skladištenje podataka (Pal, 2016). AVRO je format otvorenog koda namijenjen za Apache Hadoop, softversku knjižnicu i okvir za skladištenje i obradu velikih setova podataka (IBM, 2022). Među češćim formatima velikih podataka također se nalaze Parquet i ORC (engl. *Optimized Row Columnar*). Iako dijele neka obilježja koja ima i AVRO, najveća razlika proizlazi iz načina kako pohranjuju podatke (Woodie, 2018). Parquet i ORC su stupčasti formati, koji zbog takvog načina pohrane smanjuju opseg zahtjeva i količinu podataka koja se učitava s diska.

Slika 1. Način pohrane podataka u redčanom i stupčastom formatu



Izvor: prilagođeno prema Getz, 2014

6. 2 Kompresija podataka

Kompresija podataka još je jedna tehnika kojom je moguće smanjiti zahtjeve na ulaz i izlaz podataka. Postupak kompresije podataka može se definirati kao proces prilikom kojega se smanjuje fizički prostor potreban za pohranu podataka, korištenjem određenih metoda za zapis podataka (Farena, 2020). Prilikom korištenja resursa kao što je to slučaj kod računarstva u oblaku, kod kompresije je poželjno da podržava mogućnost dijeljenih datoteka (engl. *splittable files*). Ova mogućnost znači da se dijelovi podataka mogu obraditi na različitim računalnim čvorovima koji su neovisni o drugim čvorovima u klasteru, što može povećati brzinu i propusnost obrade (Pal, 2016). Druga mogućnost s kojom kompresija ubrzava obradu je optimiziranje veličine datoteka (Hocanin, 2017). Primjena kompresije u pravilu znači balansiranje između procesorskih računalnih kapaciteta koji su potrebni i zahtjeva za pohranom podataka. Svaki algoritam ima različita obilježja, a neki od češćih prikazani su u tablici 1.

Tablica 1. Obilježja algoritama kompresije i podrška formata podataka

Algoritam	*Splittable* mogućnost?	Vrijeme kompresije	Veličina nakon kompresije	Omjer kompresije[2]	Avro	ORC	parquet	TSV, CSV, JSON
GZIP	Ne	Srednje	Mala	Visok	Ne	Ne	Da	Da
BZIP2	Da	Sporo	Mala	Vrlo visok	Samo čitanje	Ne	Ne	Da
LZO	Da	Brzo	Srednja	Nizak	Ne	Ne	Da	Samo čitanje
Snappy	Ne	Vrlo brzo	Velika	Nizak	Samo čitanje	Da	Da	Da

Izvor: Pal, 2016; Hocanin, 2017; Amazon Web Services, 2022 e

[2] Omjer kompresije: omjer između veličine datoteke bez kompresije i veličine nakon kompresije. Što je rezultat viši, to je stupanj kompresije bolji. U pravilu, što je stupanj kompresije viši, to su veći procesorski zahtjevi (Hocanin, 2017).

Iz pregleda obilježja u tablici 1 vidljivo je kako primjerice algoritmi GZIP i BZIP2 imaju najviše omjere kompresije od prikazanih, ali s druge strane zahtijevaju duže vrijeme za obradu. LZO i Snappy su brži i rezultiraju manjim stupnjem kompresije, ali zato je primjerice Snappy podržan u najvećem broju formata od spomenutih algoritama. Primjenjivost kompresije, kao i kod formata, ovisi o poznavanju važnih obilježja i svrhe uporabe podataka.

6. 3 Particioniranje i *bucketing*

Postupci kao što su particioniranje i *bucketing* koriste se u svrhu smanjenja latencije upita. Particioniranje je postupak koji omogućava podjelu podataka tablice po jednom ili više stupaca, temeljem vrijednosti particioniranih stupaca (Pal, 2016). Prilikom provođenja standardnog upita skeniraju se svi podaci u tablici, što može usporiti izvedbu zbog količine podataka. Kada su podaci particionirani, stupac koji predstavlja particiju može se uključiti u upit, čime se skeniraju samo podaci koji pripadaju toj particiji. Idealan slučaj za particioniranje su stupci koji imaju srodne vrijednosti ograničenog raspona, primjerice podaci o odjelima u organizaciji, država ili godine. U tom kontekstu važna je kardinalnost podataka (jedinstvenost vrijednosti podataka stupca), na način da je particioniranje primjenjivije kod manje kardinalnosti gdje stupac ima ograničen broj jedinstvenih vrijednosti (Amazon Web Services, 2022 f).

Bucketing je postupak kojime se postiže strukturiranje podataka stupaca u dijelove kojima je lakše upravljati, a sastoji se od stvaranja segmenata koji sadrže dio seta podataka (Pal, 2016). Ovaj postupak najprimjenjiviji je kada se provodi nad stupcem koji ima visoku kardinalnost (velik broj jedinstvenih vrijednosti) i ujednačenu raspršenost vrijednosti u setu (Amazon Web Services, 2022 f). Za ovaj postupak najbolje je odabrati stupac koji nema puno *null* vrijednosti, čiji se podaci mogu podijeliti u jednolike skupine, a skupine sadrže sličnu količinu podataka (Amazon Web Services, 2022 f). Primjeri takvih atributa su identifikatori ili vremenske oznake događaja.

6. 4 Struktura upita i odabir naredbi

SQL upiti koji su ne samo funkcionalni, već i prilagođeni da budu efikasniji (engl. *query tuning*) mogu znatno poboljšati performanse izvedbe. Neki od postupaka koji se mogu primijeniti kroz strukturu i odabir naredbi opisani su u nastavku.

1) Izbjegavanje SELECT * klauzule. Budući da se ovom klauzulom dohvaćaju svi stupci tablice, uporaba ovog izraza može usporiti izvođenje upita (Mehrotra, 2016). Iako korisniku može biti jednostavnija za korištenje, poželjno je kod svakog upita razmotriti koji stupci su zapravo potrebni i dohvatiti samo njih.

2) JOIN naredbe. JOIN spojevi tablica mogu se izvoditi koristeći više vrsta spojeva ovisno o potrebnom ishodu, a čije specifičnosti su cijela zasebna tema. Međutim, neke od općih smjernica korištenja JOIN naredbi u kontekstu optimizacije upita uključuju (Schwartz, Zaitsev i Tkachenko, 2012):

a) Redoslijed tablica. Određivanje redoslijeda kojim se tablice spajaju u pojedinom JOIN spoju može značajno utjecati na performanse upita.

b) Odabir vrste spoja. Ovisno o željenom ishodu, važno je razmotriti mogu li se zahtjevi za spajanjem optimizirati korištenjem druge vrste spoja (npr. unutarnjeg spoja umjesto vanjskog), sužavanjem broja tablica i načina spajanja te poznavanjem obilježja (npr. indeksa) atributa kojima se tablice spajaju.

3) GROUP BY naredba. Ova naredba koja se koristi za grupiranje podataka može dovesti do većih zahtjeva resursa u slučaju da se ne koristi optimalno. Postupak koji može poboljšati njezino korištenje odnosi se na ograničavanje broja stupaca u SELECT naredbi, čime se smanjuju memorijski zahtjevi za skladištenje podataka o redovima koji se ovom naredbom grupiraju (Hocanin, 2017).

4) ORDER BY naredba. Ova naredba za uzlazno ili silazno sortiranje podataka izvršava se nad svim redcima upita za koji je definirana, zbog čega može predstavljati značajan memorijski zahtjev kod većeg skupa podataka (Hocanin, 2017). Za dohvaćanje najviših ili najnižih vrijednosti nekog atributa postoje efikasnije naredbe koje mogu zamijeniti ili suziti ORDER BY sortiranje ograničavanjem broja podataka.

5) Ograničavanje broja redaka i vrijednosti. U slučajevima kada je potrebno prikazati samo određeni broj redaka, moguće je koristiti naredbe LIMIT ili TOP (Mujadžević, 2016). Primjerice, ako korisnik želi prikazati samo podskup podataka od 500 redaka, može to postići navođenjem LIMIT 500. Naredba TOP navodi se u SELECT izrazu i dohvaća zadani broj prvih redaka tablice s vrijednostima odabranih atributa.

6) Korištenje LIKE klauzule. Budući da tekstualne vrijednosti mogu sadržavati različite varijante sadržaja, nerijetko se događa da korisnik ima potrebu pretražiti vrijednost nekog stupca tako da se ona samo djelomično preklapa sa zadanim znakovnim nizom. Međutim, korištenje ovog postupka u pretraživanju može biti zahtjevno sa strane potrebnih resursa, posebice kada je potrebno uključiti više LIKE uvjeta u pretragu. Koristeći Presto, višestruki LIKE uvjeti mogu se zamijeniti funkcijom *regexp_like*, koja zbog drugačije metode pretraživanja obrasca zahtijeva manje resursa (Medium, 2021).

Osim navedenih postupaka, na temu *query tuning* mogućnosti mogao bi se navesti još cijeli niz opcija kojima se mogu poboljšati rezultati upita. Do sada opisane mogućnosti predstavljaju neke od češće primjenjivanih postupaka, ali i elemente koji se primjenjuju ili su relevantni za sljedeći dio rada.

7. REZULTATI OPTIMIZACIJE PODATAKA I SQL UPITA NA PRIMJERU U AWS SERVISIMA

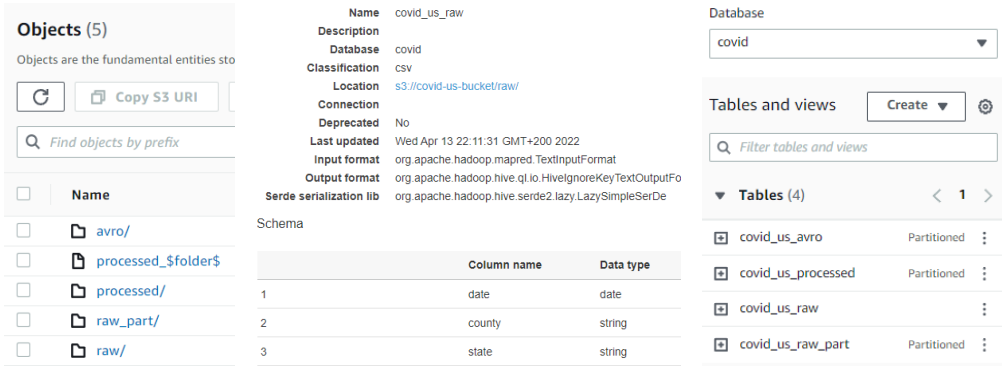
Podaci korišteni za postupke u nastavku preuzeti su iz Kaggle setova podataka (Davis, 2021). Korišteni set uključuje podatke o COVID-19 slučajevima, socioekonomskim obilježjima i vremenskim prilikama u Sjedinjenim Američkim Državama tijekom 2020. godine. Podaci su preuzeti kao .csv dokument veličine 1.3 GB, a sastoje se od 227 stupaca različitih tipova podataka i ukupno 790331 redaka. U ovom radu, navedeni set podataka korišten je za kreiranje četiri relacijske

tablice s različitim podacima u AWS servisima, a nad kojima su provedeni SQL upiti i prikupljeni rezultati provedbe.

7. 1 Kreiranje tablica i obilježja podataka

Izvorni podaci najprije su učitani kao objekt u S3 spremnik pod prefiks[3] *raw*. Na slici 2 (lijevo) prikazani su objekti S3 spremnika s prefiksima nakon kreiranja svih tablica. U AWS Glue servisu zatim je kreiran prvi *crawler* kojim je dobivena shema podataka vidljiva u Glue Data Catalogu (u tablici covid_us_raw), a kojoj je dio podataka prikazan na slici 2 (sredina). Stvaranjem tablice u Glue Data Catalogu ovi podaci postaju dostupni i u Amazon Athena servisu. Na slici 2 (desno) nalazi se prikaz Athena baze *covid* nakon kreiranja svih tablica. Koristeći originalne podatke (tablica covid_us_raw), kreirana je sljedeća tablica covid_us_raw_part. Također je u tekstualnom formatu, ali je ona kreirana korištenjem CTAS upita uz particioniranje po stupcu *state* i kompresiju koristeći GZIP algoritam. CTAS označava vrstu upita koji koristi sintaksu CREATE TABLE AS SELECT, a kojim se može kreirati nova tablica iz rezultata SELECT naredbe određenog upita (u ovom slučaju, iz *raw* podataka). Budući da je u CTAS upitu zadan i parametar kojim se podaci particioniraju po stupcu *state*, u njihovom S3 prefiksu (raw_part) kreirale su se 54 „mape" – particije. Nazivi particija odgovaraju saveznim državama, a svaka particija sadrži dokumente s podacima istoimene države.

Slika 2. Prikaz S3 spremnika s različitim prefiksima (lijevo), tablica covid_us_raw u Glue Data Catalogu (sredina) i prikaz covid baze s četiri tablice u Atheni (desno)



Izvor: autori

Treća tablica (covid_us_avro) također je kreirana korištenjem CTAS upita nad izvornim podacima, ali u AVRO formatu. Nad podacima u ovoj tablici nije provedena kompresija, a particionirani su također po stupcu *state*. Zadnja tablica (covid_us_processed) kreirana je korištenjem AWS Glue resursa koji su uključivali *job* i drugi *crawler*. Najprije je kreiran *job* putem kojega su izvorni .csv podaci iz S3 *raw* prefiksa pretvoreni u komprimirani parquet format, particionirani po stupcu *state* i pohranjeni u novi S3 prefiks *processed*. Nakon izvršenja *job* postupka, kreiran je drugi *crawler*

[3] Prefiks označava naziv koncepta mape u S3 spremniku, putem kojega se objekti mogu grupirati koristeći zajednički naziv za skupinu objekata (Amazon Web Services, 2022 a).

putem kojega je na temelju prethodno obrađenih podataka stvorena tablica covid_us_processed u Glue Data Catalogu. Sažeti prikaz obilježja pojedinih prethodno opisanih tablica i podataka koje sadrže nalazi se u tablici 2.

Tablica 2. Pregled obilježja četiri kreirane tablice

	Postupak	Format	Kompresija	Particije
covid_us_raw	AWS Glue	CSV	-	-
covid_us_avro	CTAS	AVRO	-	stupac *state*
covid_us_raw_part	CTAS	CSV	GZIP	stupac *state*
covid_us_processed	AWS Glue	PARQUET	GZIP	stupac *state*

Izvor: autori

7. 2 Korišteni SQL upiti

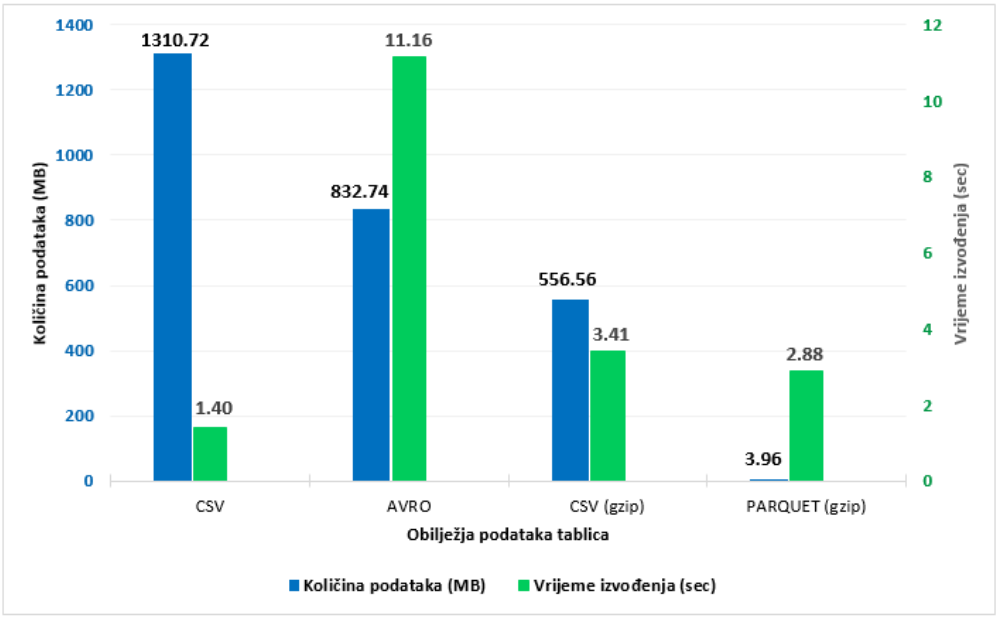
Za prikupljanje rezultata ukupno je korišteno osam SQL upita (izrada autora). Prva dva upita primijenjena su u svrhu usporedbe rezultata ovisno o formatu, kompresiji i particioniranju podataka, dok su preostali upiti inačice u nekoliko primjera temeljem kojih su dobiveni pokazatelji za usporedbu upita ovisno o stupnju optimiziranosti. Prvi upit (Prilog, upit 1) primijenjen je u svrhu prikaza rezultata (količine skeniranih podataka i vremena izvođenja upita) nad cijelim setovima podataka, u tablicama koje se razlikuju po formatu i postupku kompresije podataka. Upitom se dobivaju podaci o prvih pet saveznih država sa svojim vodećim okrugom po postotku zabilježenih COVID-19 slučajeva u odnosu na ukupnu populaciju okruga. Drugi upit (Prilog, upit 2) korišten je u svrhu dobivanja rezultata iz kojih je vidljiv učinak particioniranja, putem WHERE uvjeta koji uključuje odabrane savezne države. Ovim upitom dobivaju se podaci koji prikazuju najveći postotak cijepljenih stanovnika u pet prvih država i njihovih vodećih okruga po zabilježenom broju slučajeva u odnosu na populaciju, počevši od najmanje procijepljenosti. Treći upit (Prilog, upit 3) ima dvije srodne inačice koje se razlikuju po stupnju optimiziranosti, ali vraćaju iste rezultate: prikazuju podatke zabilježene tijekom ljetnih mjeseci 2020. Preostali upiti (Prilog, upit 4) predstavljaju četiri srodne verzije istog upita koje se razlikuju po tome što uključuju ili isključuju pojedine naredbe. Upitima se dobivaju vremenski podaci na razini saveznih država o tome kada je u 2020. godini zabilježen prvi i zadnji COVID-19 slučaj, koliki je raspon dana u kojemu su bilježeni slučajevi te koji je zadnji dostupan omjer slučajeva u populaciji na zadnji datum podataka.

7. 3 Rezultati provedenih SQL upita

Korišteni upiti provedeni su u Amazon Athena servisu. Količina skeniranih podataka u istom upitu nije promjenjiva, ali vrijeme izvedbe jest, zbog čega se kao rezultat vremena izvođenja upita računao prosjek pet izvedbi. Dobiveni rezultati prikazani su u tri skupine: 1) rezultati prvog upita za usporedbu formata podataka i postupaka kompresije, 2) rezultati drugog upita koji prikazuju učinak particioniranja i 3) usporedba performansi upita u odnosu na stupanj optimizacije.

1) Usporedba formata podataka i postupaka kompresije. Za ovu usporedbu korišteni su rezultati dobiveni prvim upitom (Prilog, upit 1), prikazani na grafikonu 1. Ovaj upit skenira cijeli set podataka u tablicama. Stupci koji su u grafikonu prikazani u skupinama nazvanima prema formatu i algoritmu kompresije odnose se na četiri kreirane tablice koje sadrže podatke s navedenim obilježjima.

Grafikon 1. Prikaz performansi prvog SQL upita nad podacima četiri različite tablice



Izvor: autori

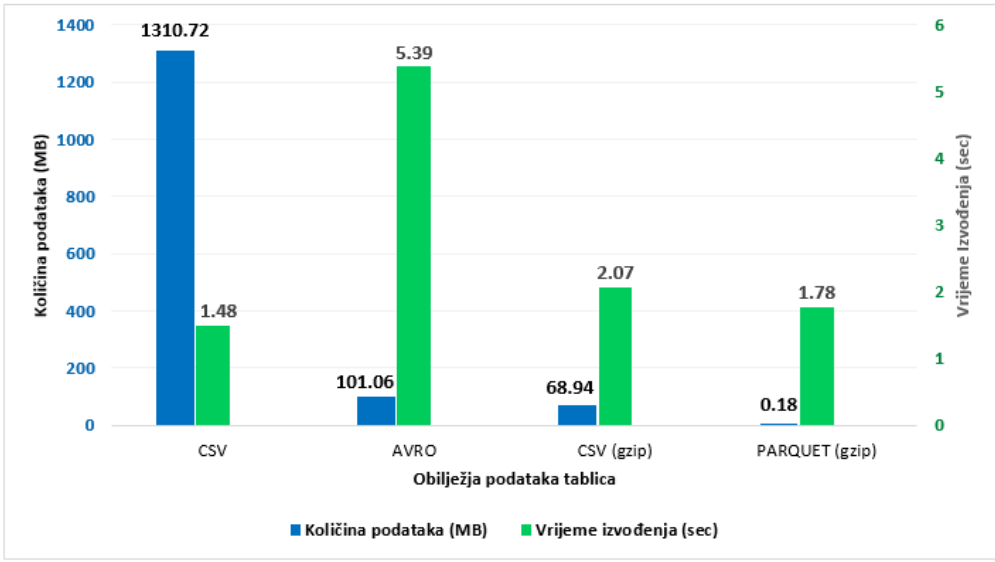
Iz grafikona 1 vidljivo je kako je najkraće vrijeme izvedbe u prosjeku zabilježeno u provedbi upita kod CVS formata bez kompresije (izvorni podaci), ali istovremeno je u ovoj tablici zabilježena najveća količina skeniranih podataka (1310,72 MB). U usporedbi s ovom tablicom, provedbom istog upita nad AVRO podacima bez kompresije trajanje izvedbe upita bilo je znatno duže – gotovo 8 puta više vremena, ali je količina skeniranih podataka bila manja (832,74 MB). Ovi rezultati u skladu su primjerice s odnosom koji je ispitivao Radečić (2021) uspoređujući performanse CSV i AVRO formata, u kojemu je također pronašao kako AVRO zahtijeva duže vrijeme od CSV-a kod čitanja podataka (engl. *read time*), a reducira veličinu podataka. Zadnja skupina na grafikonu 1 prikazuje rezultate provedbe upita nad tablicom s parquet podacima. Budući da su podaci u ovoj tablici komprimirani, mogu se usporediti s kategorijom CSV (gzip) koja također sadrži komprimirane podatke, ali u CSV formatu. Vidljivo je kako se provedbom istog upita nad parquet podacima količina skeniranih podataka izrazito smanjila (na 3,96 MB). U odnosu na izvorne podatke, na parquet podacima skenirana je čak 330 puta manja količina podataka, dok je izvedba upita bila približno 15 % brža nego kod komprimiranog CSV-a.

U odnosu na izvorne CSV podatke bez kompresije, nad komprimiranim podacima istog formata upitom je skenirano 58 % manje podataka (556,56 MB u odnosu na početnih 1310,72 MB). Iz ovog

odnosa vidljiv je značajan doprinos gzip algoritma kompresije, a najveća razlika primjetna je u slučaju parquet formata koji prema osnovnim postavkama primjenjuje gzip kompresiju. Općenito se u dobivenim rezultatima parquet pokazao najefikasnijim formatom za pohranu i skeniranje podataka. Budući da količina skeniranih podataka u Atheni izravno utječe na troškove upita, korištenje parquet podataka u ovom slučaju je i najpovoljnija opcija. Zbog toga, na većoj količini podataka planiranje odabira formata i postupaka nad podacima može imati značajan utjecaj na komponente troškova.

2) Particioniranje. Rezultati ovog postupka prikazani su na grafikonu 2, a predstavljaju vrijednosti dobivene provedbom drugog upita (Prilog, upit 2). U ovom slučaju, upit je strukturiran na način da u WHERE uvjetu koristi stupac *state*, a koji u pojedinim tablicama predstavlja atribut po kojemu su podaci particionirani.

Grafikon 2. Prikaz performansi drugog SQL upita nad podacima četiri različite tablice



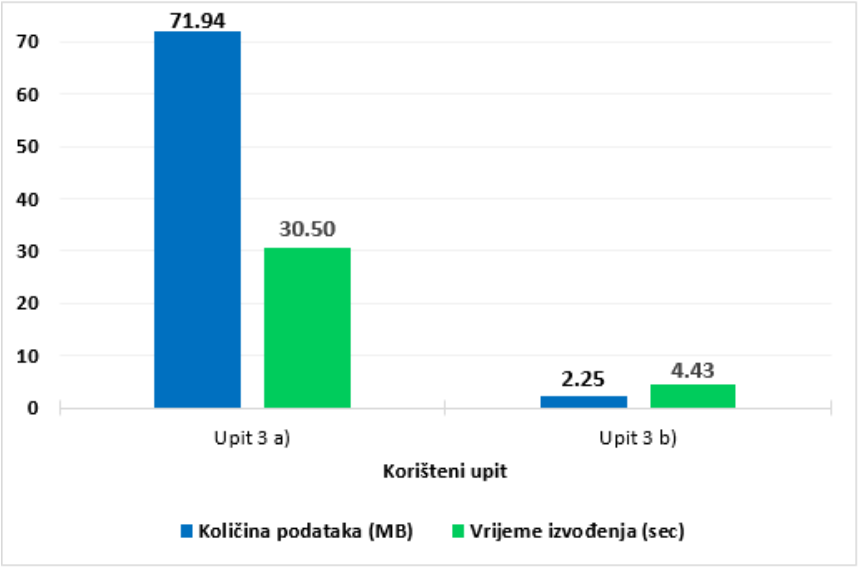
Izvor: autori

Tablica s izvornim podacima (CSV) nije particionirana, zbog čega se neovisno o uvjetu skenira cijeli set podataka (1310,72 MB). Podaci u svim ostalim tablicama particionirani su po stupcu *state*. U usporedbi s izvornim podacima, nad drugom tablicom koja sadrži CSV podatke (CSV gzip) drugi upit skenira značajno manje podataka (68,94 MB). Ako usporedimo ovaj rezultat s istom tablicom i njezinim vrijednostima prvog upita koji se izvršio nad cijelim setom komprimiranih podataka CSV tablice (grafikon 1), vidljivo je kako se korištenjem particija količina skeniranih podataka znatno smanjila (nad cijelom tablicom skenira se 556,56 MB). Isti odnos vidljiv je i kod podataka ostalih tablica: iz grafikona 2 primjetno je kako u odnosu na izvorne CSV podatke, sve ostale particionirane tablice skeniraju znatno manje podataka. Ovaj odnos vidljiv je i iz usporedbe vrijednosti za iste tablice na grafikonu 1 i grafikonu 2: u drugom upitu s WHERE uvjetom koji uključuje pet saveznih država, tablica s AVRO podacima skenira 88 % manje podataka (101,06 MB naspram 832,74 MB),

dok se količina skeniranih podataka parquet formata smanjila za 95 % (0,18 MB u odnosu na 3,96 MB). Kako se korištenjem particija smanjila količina skeniranih podataka, tako se smanjilo i prosječno vrijeme izvedbe upita.

3) Usporedba performansi u odnosu na stupanj optimizacije upita. Na grafikonu 3 prikazani su rezultati dobiveni provedbom dvije inačice trećeg SQL upita (Prilog, upit 3) nad podacima u tablici covid_us_processed.

Grafikon 3. Prikaz performansi inačica trećeg SQL upita na parquet podacima



Izvor: autori

U prvoj inačici upita (Prilog, upit 3a) naredbom SELECT odabiru se svi stupci (SELECT *). WHERE uvjetom dohvaćaju se podaci samo za određene mjesece, navođenjem niza LIKE klauzula s logičkim operatorom OR. U drugoj inačici upita (Prilog, upit 3b) uvedene su dvije izmjene s ciljem bolje optimizacije upita, a uključuju odabir samo potrebnih stupaca (njih ukupno 7) i zamjenu niza LIKE klauzula funkcijom *regexp_like*. Rezultati dobiveni provedbom inačice 3a pokazuju kako se neoptimiziranim upitom na parquet formatu skenira 71,94 MB podataka. Za provedbu ovog upita prosječno je bilo potrebno 30,5 sekundi. U usporedbi s njim, optimizirani upit skenira znatno manje podataka (2,25 MB), zbog čega je i vrijeme provedbe upita gotovo 7 puta brže. Najveći udio smanjenja količine skeniranih podataka u ovoj usporedbi odnosi se na odabir nekolicine stupaca koji daju potrebne podatke, uz što se mogu primijeniti i drugi postupci za dodatno poboljšanje performansi.

Rezultati prikazani na grafikonu 4 odnose se na posljednji primjer u kojemu su uspoređene inačice upita koje se razlikuju po određenim naredbama putem kojih se postiže manji ili veći stupanj optimiziranosti. Četiri inačice SQL upita (Prilog, upit 4) provedene su nad parquet podacima u tablici covid_us_processed. Razlike među inačicama su sljedeće:

4 a) uključuje ORDER BY naredbu;

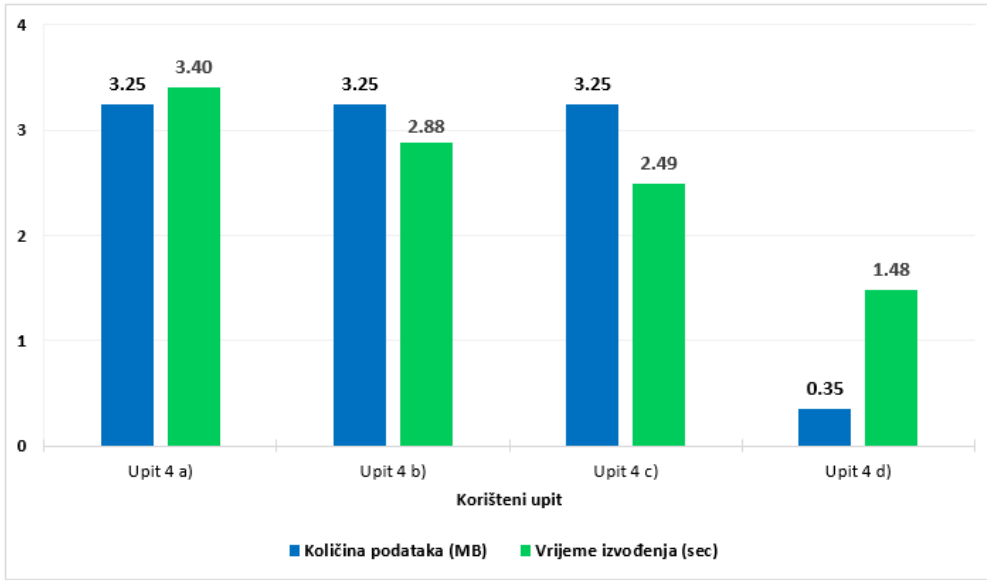
4 b) uključuje ORDER BY i LIMIT naredbe;

4 c) izostavlja ORDER BY i LIMIT naredbe;

4 d) izostavlja ORDER BY i LIMIT naredbe te uključuje WHERE uvjet za dohvaćanje podataka tri odabrane savezne države.

Prve tri inačice upita dohvaćaju iste podatke, a razlikuju se prema naredbi za sortiranje i/ili ograničavanje (limitiranje) podataka na kraju upita. Iz grafikona 4 vidljivo je kako skeniraju jednaku količinu podataka (3,25 MB), dok se prosječno vrijeme izvođenja razlikuje. Najduže vrijeme izvođenja (3,40 sec) zabilježeno je u inačici 4a (Prilog, upit 4a), koja koristi ORDER BY naredbu nad cijelim setom podataka, iz čega je vidljivo kako već samo ova naredba može usporiti izvođenje upita. U usporedbi s ovom inačicom, u 4b (Prilog, upit 4b) dodatno je uključena naredba LIMIT kojom se ograničava dohvaćanje 10 sortiranih redaka. Time se vrijeme izvođenja upita smanjilo za 15 % (2,88 u odnosu na 3,40 sekundi). U trećoj inačici (Prilog, upit 4c) izostavljena je naredba za sortiranje, čime se vrijeme izvođenja dodatno smanjilo na prosječno 2,49 sekunde. Najveća razlika u inačicama prisutna je u zadnjoj varijanti upita (Prilog, upit 4d), u kojoj se iz podataka uzimaju samo države za čije rezultate smo zainteresirani (u ovom slučaju tri najmnogoljudnije savezne države: Kalifornija, Teksas i Florida). Ograničavanjem broja podataka kroz stupac *state*, po kojemu su podaci particionirani, upit skenira znatno manju količinu podataka (0,35 MB u odnosu na prijašnjih 3,25 MB), a time je i vrijeme izvođenja upita kraće.

Grafikon 4. Prikaz performansi inačica četvrtog SQL upita na parquet podacima



Izvor: autori

Iz ovog primjera vidljivo je kako se uključivanjem samo jedne dodatne naredbe (ORDER BY) vrijeme izvođenja upita može usporiti. U situacijama u kojima je potrebno sortiranje korisno je razmotriti

opciju ograničavanja rezultata na onaj podskup koji promatramo, što se u ovom primjeru pokazalo korisnim za smanjenje vremena izvedbe upita. Poput primjera na grafikonu 3, najkorisniji postupak za optimizaciju upita bilo je ograničavanje količine dohvaćenih podataka. Osim toga, dodatni postupci poput prikazanog particioniranja mogu značajno utjecati na performanse i u konačnici trošak resursa.

8. ZAKLJUČAK

Glavni cilj ovog rada bio je prikazati, opisati i istražiti na praktičnom primjeru različita obilježja i postupke koji se mogu provesti nad podacima u svrhu optimizacije SQL upita u AWS servisima. Budući da obrada velikih podataka može zahtijevati jače računalne resurse ili veću fleksibilnost rada, zbog svojih prednosti računarstvo u oblaku postaje sve zastupljeniji model u poslovanju.

Za dohvaćanje podataka i postavljanje upita nad bazama podataka uobičajeno se koristi SQL upitni jezik, koji zbog svojih brojnih mogućnosti nudi značajan prostor za primjenu različitih postupaka u svrhu veće efikasnosti. U ključne postupke mogu se svrstati odabir formata i provedba kompresije podataka, ali i druge mogućnosti značajne za rad u oblaku, poput particioniranja podataka i planiranja strukture upita.

U praktičnom dijelu rada su putem AWS servisa kreirane četiri tablice različitih obilježja. Glavna obilježja po kojima se razlikuju obuhvaćaju format podataka (csv/avro/parquet), odabir kompresije (bez kompresije/gzip) i primjenu postupka particioniranja (da/ne). Rezultati su prikupljeni na način da je nad ovim podacima proveden veći broj SQL upita, a zatim su uspoređene dobivene vrijednosti. Po pitanju formata podataka, najučinkovitijim se pokazao parquet, dok se provođenje postupka kompresije općenito pokazalo korisnim za smanjenje količine skeniranih podataka. Postupak particioniranja pokazao je prednosti u svim primijenjenim slučajevima. Sljedeći dio rezultata uključivao je postupke optimizacije strukture SQL upita i odabira naredbi. Neki od postupaka koji su primijenjeni, a koji su pokazali da se njima može poboljšati izvedba upita, uključuju planiranje i dohvat samo potrebnih stupaca, primjenu odgovarajućih funkcija, pragmatično korištenje sortiranja, primjenu ograničenja broja rezultata i prilagodbu upita za uporabu particija. Temeljem opisanih postupaka i dobivenih rezultata, može se zaključiti kako postoji veći broj mogućnosti putem kojih se može optimizirati obrada podataka SQL upitima, a koji su važni jer doprinose učinkovitom i isplativom korištenju usluga u oblaku za rad s velikim podacima.

LITERATURA

Amazon Web Services (2022 a) *Amazon Simple Storage Service – User Guide*. Dostupno na: https://docs.aws.amazon.com/AmazonS3/latest/userguide/s3-userguide.pdf (pristupljeno: 23.3.2022.)

Amazon Web Services (2022 b) *Amazon Athena – User Guide*. Dostupno na: https://docs.aws.amazon.com/athena/latest/ug/athena-ug.pdf (pristupljeno: 24.3.2022.)

Amazon Web Services (2022 c) *AWS Glue Developer Guide*. Dostupno na: https://docs.aws.amazon.com/glue/latest/dg/glue-dg.pdf (pristupljeno: 26.3.2022.)

Amazon Web Services (2022 d) *What is AWS*. Dostupno na: https://aws.amazon.com/what-is-aws/ (pristupljeno: 20.3.2022.)

Amazon Web Services (2022 e) *Athena Compression Support*. Dostupno na: https://docs.aws.amazon.com/athena/latest/ug/compression-formats.html (pristupljeno: 6.4.2022.)

Amazon Web Services (2022 f) *Bucketing vs Partitioning*. Dostupno na: https://docs.aws.amazon.com/athena/latest/ug/bucketing-vs-partitioning.html (pristupljeno: 9.4.2022.)

Ashari, R., Akbar, M. F., Thamrin, W. D., Hanafiah, N. (2021) "A Systematic Literature Review: Database Optimization Techniques", *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)*, str. 295-300, DOI: 10.1109/ICCSAI53272.2021.9609766

BasuMallick, C. (2022) *AWS vs. Azure: Understanding the Key Differences*. Dostupno na: https://www.spiceworks.com/tech/cloud/articles/aws-vs-azure/ (pristupljeno: 7.11.2022.)

Buyya, R., Calheiros, R. N., Dastjerdi, A. V. (Ur.) (2016) *Big Data Principles and Paradigms*, Cambridge, MA: Elsevier Inc.

Data Semantics (2021) *AWS vs Azure – A Detailed Comparison*. Dostupno na: https://datasemantics.co/aws-vs-azure-cloud-platforms (pristupljeno: 7.11.2022.)

Davis, J. (2021) *US Counties: COVID19 + Weather + Socio/Health data*. Dostupno na: https://www.kaggle.com/datasets/johnjdavisiv/us-counties-covid19-weather-sociohealth-data?select=US_counties_COVID19_health_weather_data.csv (pristupljeno: 13.4.2022.)

Dumbill, E. (2012) *Planning for Big Data*, Sebastopol, CA: O'Reilly Media Inc.

Farena, N. (2020) *Kompresija podataka*, diplomski rad, Osijek: Sveučilište J. J. Strossmayera u Osijeku

Gartner Glossary (2022) *Cloud Computing*. Dostupno na: https://www.gartner.com/en/information-technology/glossary/cloud-computing (pristupljeno: 20.3.2022.)

Getz, A. (2014) *Column And Row Based Database Storage*. Dostupno na: https://bi-insider.com/business-intelligence/column-and-row-based-database-storage/ (pristupljeno: 4.4.2022.)

Gillis, A. S. (2020) *Amazon Web Services (AWS)*. Dostupno na: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services (pristupljeno: 23.3.2022.)

Hocanin, M. (2017) *Top 10 Performance Tuning Tips for Amazon Athena*. Dostupno na: https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/ (pristupljeno: 6.4.2022.)

IBM (2022) *Apache Avro*. Dostupno na: https://www.ibm.com/topics/avro (pristupljeno: 4.4.2022.)

IBM Cloud (2020) *Top 7 Most Common Uses of Cloud Computing*. Dostupno na: https://www.ibm.com/cloud/blog/top-7-most-common-uses-of-cloud-computing (pristupljeno: 20.3.2022.)

Jain, N., Raghu, B., Khanna, V. (2021) „Query Optimization for Cloud Database", *2021 6th International Conference for Convergence in Technology (I2CT)*, str. 1-3, DOI: 10.1109/I2CT51068.2021.9417911

Krmpotić, G. (2020) *SaaS VS PaaS VS IaaS*. Dostupno na: www.gorankrmpotic.eu/saas-vs-paas-vs-iaas/ (pristupljeno: 20.3.2022.)

Le, Q. H., Xie, J., Millington, D., Waniss, A. (2015) „Comparative Performance Analysis of PostgreSQL High Availability Database Clusters through Containment", *International Journal of Advanced Research in Computer and Communication Engineering*, 4 (12), str. 526-533. DOI: 10.17148/IJARCCE.2015.412150

Liu, Y., Guo, S., Hu, S., Rabl, T., Jacobsen, H., Li, J., Wang, J. (2018) „Performance Evaluation and Optimization of Multi-Dimensional Indexes in Hive", *IEEE Transactions on Services Computing*, 11 (5), str. 835-849, DOI: 10.1109/TSC.2016.2594778

Medium (2021) *10 Tips For Presto Query Performance Optimization*. Dostupno na: https://medium.com/@seals_xyz/10-tips-for-presto-query-performance-optimization-37fa0b7c6dc3 (pristupljeno: 10.4.2022.)

Mehrotra, S. (2016) *SQL Performance Tuning*. Dostupno na: https://www.globallogic.com/il/wp-content/uploads/2016/02/SQL-Performance-Tuning.pdf (pristupljeno: 10.4.2022.)

Michels, J., Hare, K., Kulkarni, K., Zuzarte, C., Liu, Z. H., Hammerschmidt, B., Zemke, F. (2018) „The New and Improved SQL: 2016 Standard", *ACM SIGMOD Record*, 47(2), str. 51-60. DOI: 10.1145/3299887.3299897

Mujadžević, E. (2016) *Uvod u SQL – priručnik za polaznike*, Zagreb: Sveučilišni računski centar

Nerić, V., Sarajlić, N. (2020) „A Review on Big Data Optimization Techniques", *B&H Electrical Engineering*, 14 (2), str. 13-18. DOI: 10.2478/bhee-2020-0008

Pal, S. (2016) *SQL on Big Data: Technology, Architecture and Innovation*, Wilmington, MA: Apress

PCSC (2018) *Public Cloud Services Comparison*. Dostupno na: https://comparecloud.in/ (pristupljeno: 20.3.2022.)

ProjectPro (2022) *AWS vs Azure-Who is the big winner in the cloud war*? Dostupno na: https://www.projectpro.io/article/aws-vs-azure-who-is-the-big-winner-in-the-cloud-war/401 (pristupljeno: 7.11.2022.)

Radečić, D. (2021) *CSV Files for Storage? Absolutely Not. Use Apache Avro Instead*. Dostupno na: https://towardsdatascience.com/csv-files-for-storage-absolutely-not-use-apache-avro-instead-7b7296149326 (pristupljeno: 23.4.2022.)

Rösch, M. (2016) „Selecting the Right Cloud Service(s)", *University of Zurich*, DOI: 10.13140/RG.2.1.5071.4000

Sander, R. (2021) *6 Ways Cloud Computing is Changing Business Today*. Dostupno na: www.smallbusinessbonfire.com/cloud-computing-changing-business-today/ (pristupljeno: 20.3.2022.)

Schwartz, B., Zaitsev, P., Tkachenko, V. (2012) *High Performance MySQL*, Sebastopol, CA: O'Reilly Media Inc.

Sellami, R., Defude, B. (2018) „Complex Queries Optimization And Evaluation Over Relational And NoSQL Data Stores In Cloud Environments", *IEEE Transactions on Big Data*, 4(2), str. 217-230. DOI: 10.1109/TBDATA.2017.2719054

Sharma, M., Kaur, J. (2020) „Performance Analysis of Queries with Hive Optimized Data Models". U: Singh, P., Kar, A., Singh, Y., Kolekar, M., Tanwar, S. (Ur.) *Proceedings of ICRIC 2019, Lecture Notes in Electrical Engineering*, 597. Springer, Cham. DOI: 10.1007/978-3-030-29407-6_49

Statista Research Department (2022) *Big data – Statistics & Facts*. Dostupno na: https://www.statista.com/topics/1464/big-data/#dossierKeyfigures (pristupljeno: 26.3.2022.)

Vailshery, L. S. (2022) *Organizations' use of cloud vendors worldwide 2021 by vendor*. Dostupno na: www.statista.com/statistics/1224552/organization-usecloud-provider-global (pristupljeno: 20.3.2022.)

Vukelić, D. (2014) *Izrada ETL alata za transformaciju podataka iz polustrukturiranih izvora*, diplomski rad, Varaždin: Fakultet organizacije i informatike

Woodie, A. (2018) *Big Data File Formats Demystified*. Dostupno na: https://www.datanami.com/2018/05/16/big-data-file-formats-demystified/ (pristupljeno: 4.4.2022.)

PRILOG: SQL UPITI

Upit 1

```
WITH cases_data AS (
SELECT state,
       county,
       ROUND((CAST(cases AS double)/
       total_population * 100), 2)
       AS cases_population_perc
FROM covid_us_*)
, worst_counties AS (
SELECT *,
      ROW_NUMBER() OVER
      (PARTITION BY state ORDER BY
cases_population_perc DESC) AS rownm
FROM cases_data)
SELECT state,
       county,
       cases_population_perc
FROM worst_counties
WHERE rownm = 1
ORDER BY cases_population_perc DESC
LIMIT 5
```

Rezultat provedbe upita 1:

#	state	county	cases_population_perc
1	Tennessee	Trousdale	24.2
2	Colorado	Crowley	23.66
3	Kansas	Norton	20.19
4	South Dakota	Bon Homme	19.87
5	Arkansas	Lincoln	18.31

Upit 2

```
SELECT state,
       county,
       MAX(percent_vaccinated)
       AS perc_vaccinated
FROM covid_us_*
WHERE (state = 'Tennessee' OR
       state = 'Colorado' OR
       state = 'Kansas' OR
       state = 'South Dakota' OR
       state = 'Arkansas')
AND state||county IN ('TennesseeTrousdale',
'ColoradoCrowley', 'KansasNorton',
'South DakotaBon Homme','ArkansasLincoln')
GROUP BY state, county
ORDER BY perc_vaccinated
```

Rezultat provedbe upita 2:

#	state	county	perc_vaccinated
1	South Dakota	Bon Homme	33.0
2	Colorado	Crowley	39.0
3	Kansas	Norton	39.0
4	Arkansas	Lincoln	42.0
5	Tennessee	Trousdale	58.0

Upit 3 (dvije inačice)

```
-- a)
SELECT *
FROM covid_us_processed
WHERE CAST(date AS varchar) LIKE '%-06-%' OR
      CAST(date AS varchar) LIKE '%-07-%' OR
      CAST(date AS varchar) LIKE '%-08-%'
ORDER BY state, county, date
```

```
-- b)
SELECT date,
       state,
       county,
       cases,
       deaths,
       percent_vaccinated,
       percent_uninsured
FROM covid_us_processed
WHERE regexp_like (CAST(date AS varchar),
'-06-|-07-|-08-')
ORDER BY state, county, date
```

Dio rezultata provedbe upita 3b:

#	date	state	county	cases	deaths	percent_vaccinated	percent_uninsured
1	2020-06-01	Alabama	Autauga	234	5	41.0	8.7216859511
2	2020-06-02	Alabama	Autauga	240	5	41.0	8.7216859511
3	2020-06-03	Alabama	Autauga	240	5	41.0	8.7216859511
4	2020-06-04	Alabama	Autauga	242	5	41.0	8.7216859511

Upit 4 (četiri inačice)

```
-- a)
WITH cases_data AS (
SELECT state
     , cases
     , date
     , total_population
     , FIRST_VALUE(date) OVER (PARTITION BY
state ORDER BY date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS first_date
     , LAST_VALUE(date) OVER
(PARTITION BY state ORDER BY date ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING)
AS last_date
FROM covid_us_processed)
SELECT state
     , first_date
     , last_date
     , ABS(DATE_DIFF ('day',
from_iso8601_date(last_date),
from_iso8601_date(first_date))) AS dates_range
     , ROUND(CAST(SUM(cases) AS
double)/SUM(total_population)*100, 2) AS
latest_cases_population_perc
FROM cases_data
WHERE date = last_date
GROUP BY state, first_date, last_date
ORDER BY dates_range DESC
```

```
-- b)
WITH cases_data AS (
SELECT state
     , cases
     , date
     , total_population
     , FIRST_VALUE(date) OVER (PARTITION BY
state ORDER BY date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS first_date
     , LAST_VALUE(date) OVER
(PARTITION BY state ORDER BY date ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING)
AS last_date
FROM covid_us_processed)
SELECT state
     , first_date
     , last_date
     , ABS(DATE_DIFF ('day',
from_iso8601_date(last_date),
from_iso8601_date(first_date))) AS dates_range
     , ROUND(CAST(SUM(cases) AS
double)/SUM(total_population)*100, 2) AS
latest_cases_population_perc
FROM cases_data
WHERE date = last_date
GROUP BY state, first_date, last_date
ORDER BY dates_range DESC
LIMIT 10
```

```
-- c)
WITH cases_data AS (
SELECT state
     , cases
     , date
     , total_population
     , FIRST_VALUE(date) OVER (PARTITION BY
state ORDER BY date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS first_date
     , LAST_VALUE(date) OVER
(PARTITION BY state ORDER BY date ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING) AS
last_date
FROM covid_us_processed)
SELECT state
     , first_date
     , last_date
     , ABS(DATE_DIFF ('day',
from_iso8601_date(last_date),
from_iso8601_date(first_date))) AS dates_range
     , ROUND(CAST(SUM(cases) AS
double)/SUM(total_population)*100, 2) AS
latest_cases_population_perc
FROM cases_data
WHERE date = last_date
GROUP BY state, first_date, last_date
```

```
-- d)
WITH cases_data AS (
SELECT state
     , cases
     , date
     , total_population
     , FIRST_VALUE(date) OVER (PARTITION BY
state ORDER BY date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS first_date
     , LAST_VALUE(date) OVER
(PARTITION BY state ORDER BY date ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING) AS
last_date
FROM covid_us_processed
WHERE state IN ('California','Texas','Florida'))
SELECT state
     , first_date
     , last_date
     , ABS(DATE_DIFF ('day',
from_iso8601_date(last_date),
from_iso8601_date(first_date))) AS dates_range
     , ROUND(CAST(SUM(cases) AS
double)/SUM(total_population)*100, 2) AS
latest_cases_population_perc
FROM cases_data
WHERE date = last_date
GROUP BY state, first_date, last_date
```

Dio rezultata provedbe upita 4a:

#	state	first_date	last_date	dates_range	latest_cases_population_perc
1	Washington	2020-01-21	2020-12-04	318	2.58
2	Illinois	2020-01-24	2020-12-04	315	6.0
3	California	2020-01-25	2020-12-04	314	3.41
4	Arizona	2020-01-26	2020-12-04	313	5.28

Professional paper
https://doi.org/10.31784/zvr.11.1.16
Received: 28. 6. 2021.
Accepted: 2. 12. 2022.

SQL QUERY TUNING AND OPTIMIZATION: AN EXAMPLE USING AWS

Jasmina Diković
Student, Polytechnic of Rijeka, Vukovarska 58, 51000 Rijeka, Croatia;
e-mail: jdikovic@veleri.hr

Ida Panev
PhD, Senior Lecturer, Polytechnic of Rijeka, Vukovarska 58, 51000 Rijeka, Croatia;
e-mail: ipanev@veleri.hr

ABSTRACT

Since the total amount of created data is increasing every day, requirements related to computing capacity needed to process data are also growing. Cloud computing is becoming a rapidly growing resource for data processing. Cloud usage is typically based on a pay-as-you-go model, which is the reason why different optimization possibilities represent a significant factor in performance and cost management. This paper describes and presents an example with various actions that can be performed for the purpose of optimizing data and SQL queries when using cloud services on big data. Described possibilities include choosing the appropriate data format or compression algorithm, but also other characteristics important for cloud environment, such as data partitioning and planning SQL queries structure. Presented practical example includes several SQL queries performed on differently processed data to compare performance against applied procedures and query characteristics. Based on the described features and obtained results, we can conclude there are multiple possibilities to optimize the implementation of SQL queries in cloud services. The importance of these actions arises from their contribution to the efficient and cost-effective usage of cloud services for big data.

Key words: *SQL, optimization, big data, cloud computing, Amazon Web Services*