

Fakultet tehničkih nauka  
Univerzitet u Novom Sadu

Seminarski rad

***Trino***

Student: Mentor:

Petar Trifunović E2 4/2021 Doc. dr Vladimir Dimitrieski

# Sadžaj

[1 Uvod 1](#_Toc92914725)

[2 Princip rada i osnovni koncepti *Trino* tehnologije 2](#_Toc92914726)

[2.1 Tipovi servera 2](#_Toc92914727)

[2.1.1 Koordinator 2](#_Toc92914728)

[2.1.2 Radni čvorovi 3](#_Toc92914729)

[2.2 Apstrakcija i pristup izvorima podataka 3](#_Toc92914730)

[2.2.1 Konektori 3](#_Toc92914731)

[2.2.2 Katalozi 4](#_Toc92914732)

[2.2.3 Šeme 4](#_Toc92914733)

[2.2.4 Tabele 5](#_Toc92914734)

[2.3 Model izvršavanja upita 5](#_Toc92914735)

[2.3.1 Plan upita 5](#_Toc92914736)

[2.3.2 Faze izvršavanja i fragmenti plana 5](#_Toc92914737)

[2.3.3 Zadaci 6](#_Toc92914738)

[2.3.4 *Split*-ovi 6](#_Toc92914739)

[2.3.5 Drajveri i operatori 7](#_Toc92914740)

[3 Praktični primeri primene *Trino*-a 9](#_Toc92914741)

[3.1 Katalozi 9](#_Toc92914742)

[3.1.1 Katalog za *MySQL* 9](#_Toc92914743)

[3.1.2 Katalog za *Apache Hive* 10](#_Toc92914744)

[3.2 Konfiguracija instance 10](#_Toc92914745)

[3.2.1 Konfiguracija čvora 10](#_Toc92914746)

[3.2.2 Konfiguracija servera 11](#_Toc92914747)

[3.3 *Docker-compose* fajl 11](#_Toc92914748)

[3.3.1 Servis za *Trino* 11](#_Toc92914749)

[3.3.2 Servis za *Hive* 12](#_Toc92914750)

[3.3.3 Servis za *MySQL* 12](#_Toc92914751)

[3.4 Pokretanje primera i izvršavanje upita 12](#_Toc92914752)

[4 *Presto* performanse 14](#_Toc92914753)

[4.1 *Apache Hive, Apache Spark* i *Presto* 14](#_Toc92914754)

[4.2 *Apache Spark*, *Presto* i različite verzije *Apache Hive*-a 15](#_Toc92914755)

[5 Zaključak 18](#_Toc92914756)

[6 Literatura 19](#_Toc92914757)

# Uvod

*Trino* predstavlja tehnologiju koja je inicijalno razvijena sa ciljem da omogući brži rad sa *HDFS*-om nego što je pružao *Apache Hive*. Kako je *Hive* za izvršavanje upita nad podacima koristio poseban jezik nalik na *SQL,* po imenu *HQL*, ideja jeste bila da *Trino* zadrži ovu osobinu *Hive*-a i da takođe za rad sa podacima koristi jezik visokog nivoa. Imajući u vidu to da je *SQL* već duže vreme standard kada je u pitanju zadavanje upita za rad sa podacima, potpuno je imalo smisla razviti *Trino* tako da koristi *SQL*, ali ne nad podacima u relacionim bazama. *Trino* je u početku omogućavao rad nad *HDFS*-om, ali vremenom je ova tehnologija proširena tako da podrži rad sa širokim skupom različitih izvora podataka. Tako, nastala je tehnologija koja danas podržava korišćenje *SQL*-a nad izvorima podataka za koje *SQL* inicijalno nije bio namenjen. Vrednost *Trino*-a i njemu sličnih rešenja je velika, imajući u vidu da mogućnost korišćenja opšteprihvaćenog jezika za postavljanje upita omogućava programerima, ali i ljudima drugih profesija, rad sa podacima čiji izvori ne moraju nužno biti relacione baze.

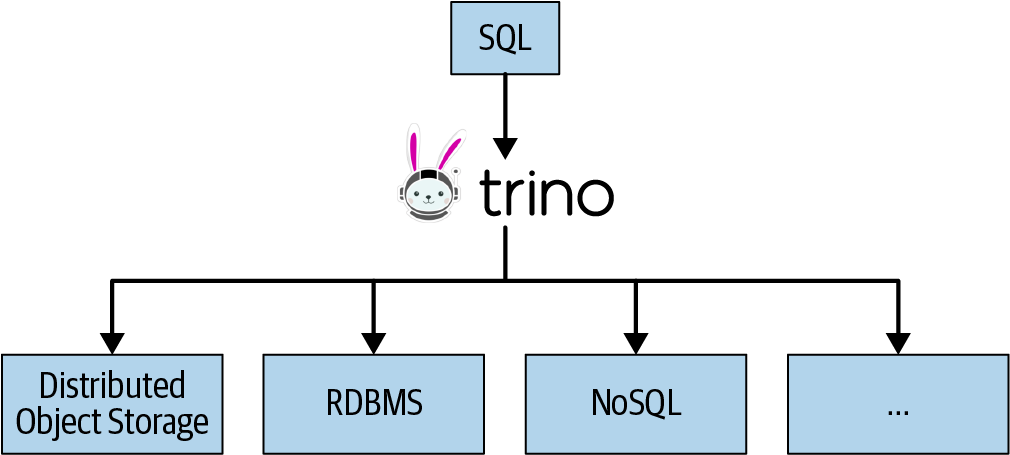
Pre početka opisivanja *Trino* tehnologije, važno je osvrnuti se na kratko na istoriju ovog softvera. 2012. godine, grupa od četvorice zaposlenih u *Facebook*-u – Martin Tavareso, Dein Sandstrom, Dejvid Filips i Erik Hvang, razvili su tehnologiju pod nazivom *Presto*. Njih četvorica zalagali su se, između ostalog, za to da *Presto* bude *open source* projekat, sa željom da se široka zajednica programera iz celog sveta uključi i doprinese razvoju ovog projekta.

2018. godine, *Facebook* je odlučio da u razvoj *Presto*-a unutar same kompanije uključi radnike koji sa *Presto* tehnologijom do tada nisu imali nikakvog iskustva, i to bez prethodnog konsultovanja sa četvoricom osnivača. Ovakav potez doveo je do toga da njih četvorica napuste *Facebook* i da nastave sa samostalnim razvojem tehnologije, proširivši joj ime u *PrestoSQL*. Ipak, i *Facebook* je nastavio sa razvojem, tako da je *Presto* nastavio da se razvija u dve nezavisne grane. *Facebook* je nakon toga pravno zaštitio naziv *Presto*, i svoju granu tehnologije nazvao *PrestoDB*. Četvorica osnivača su iz pravnih razloga bili nakon toga primorani da tehnologiju preimenuju u *Trino*. U znak poštovanja prema četvorici pokretača razvoja, u nastavku će se za ovu tehnologiju koristiti isključivo naziv *Trino*.

Ovaj rad bavi se opisivanjem *Trino* tehnologije, kako iz teorijskog, tako i iz praktičnog ugla. U poglavlju 2 dat je pregled osnovnih principa rada i najvažnijih koncepata u *Trino*-u. Nakon toga, u poglavlju 3, dat je praktičan primer korišćenja *Trino*-a za rad sa dva izvora podataka. U poglavlju 4 izvršen je pregled performansi ove tehnologije.

# Princip rada i osnovni koncepti *Trino* tehnologije

Kako je navedeno u zvaničnoj dokumentaciji [[1](#Tri)]: *„Trino je distribuirani obrađivač SQL upita, dizajniran za izvršavanje upita nad velikim skupovima podataka distribuiranim na jednom ili više heterogenih izvora podataka... Trino je (takođe) dizajniran da podrži skladišta podataka i analitiku“*. *Trino* jeste nastao kao alternativa do tada postojećim metodama za pretragu *Hadoop* fajl sistema (*HDFS*) korišćenjem jezika visokog nivoa (*Apache Hive, Apache Pig*), imajući u vidu da su te metode bile nedovoljno brze za potrebe *Facebook*-a. Ipak, *Trino* je razvijen kako bi podržao rad sa raznovrsnim izvorima podataka, a ne samo sa *HDFS*-om, nudeći mogućnost jedinstvenog pristupa tim izvorima korišćenjem *SQL*-a, kao što je prikazano na slici 2.0. U narednim poglavljima dat je pregled načina na koji *Trino* ovo omogućava.



*Slika 2.0 – Pristup različitim izvorima podataka pomoću SQL-a; preuzeto iz* [[2](#Ful21)]

Zvanična dokumentacija [[1](#Tri)], naučni rad *Presto: SQL on Everything* [[3](#Set19)], kao i knjiga koja predstavlja vodič za *Trino* [[2](#Ful21)] daju kvalitetan opis najvažnijih pojmova za razumevanje rada *Trino* tehnologije, tako da su naredna poglavlja pisana pretežno na osnovu ovih izvora.

## Tipovi servera

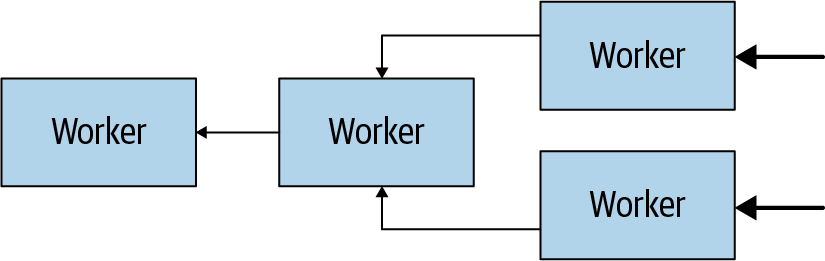
*Trino* klaster sastoji se od servera koordinatora (eng. *coordinators*) i servera radnika (eng. *workers*). U klasteru se uvek mora nalaziti jedan koordinator, uz jednog ili više radnika, a moguće je da jedna *Trino* instanca igra ulogu i koordinatora i radnika (najčešće u razvojnoj fazi).

### Koordinator

Koordinator predstavlja „glavni“ *Trino* server. On čini sponu između klijenta i *Trino* klastera, s obzirom na to da klijenti upravo ovaj čvor klastera kontaktiraju kada izdaju *SQL* iskaz[[1]](#footnote-1). Koordinator prihvata iskaz od klijenta, prevodi ga u upit, stvara plan izvršavanja i raspoređuje posao po radnim čvorovima. Takođe, na kraju, koordinator je taj koji prihvata rezultate od radnika i prezentuje ih klijentu, ili ih smešta u bafer. Koordinator je u stanju da podeli izvršavanje na faze i da prati stanje izlaznog bafera, tako da, kada klijent pročita određenu porciju rezultata iz bafera, koordinator može sam da pokrene izvršavanje naredne faze.

### Radni čvorovi

Radni čvorovi jesu oni čvorovi koji izvršavaju poslove dodeljene od strane koordinatora, i koji vrše eventualne obrade podataka. Uređeni su u stablo (slika 2.1), tako da oni na najnižem nivou, odnosno listovi stabla, vrše obradu nad podacima koji dolaze iz različitih izvora (*HDFS*, relacione baze, *NoSQL* baze...). Čvorovi na višim nivoima stabla rade sa podacima koji nastaju kao rezultat obrade čvorova na nivoima ispod njih. Radni čvorovi znaju šta treba izvršiti na osnovu zadataka koje im prethodno dodeli koordinator.



*Slika 2.1 – Primer hijerarhijskog uređenja čvorova radnika; preuzeto iz* [[2](#Ful21)]

## Apstrakcija i pristup izvorima podataka

Kako bi se omogućio jedinstven pogled na raznovrsne izvore podataka, *Trino* za njih kreira apstrakciju i iznad njih postavlja pogodan interfejs u vidu *API*-ja.

### Konektori

Konektor predstavlja komponentu apstrakcije koja povezuje *Trino* sa izvorom podataka, nezavisno od tehnologije kojom je taj izvor predstavljen (*HDFS*, relaciona baza, *NoSQL* baza...). Kako je navedeno u poglavlju 4 vodiča za *Trino* [[2](#Ful21)]: *„Dokle god je moguće podatke predstaviti u vidu tabela, kolona i vrsta korišćenjem tipova podataka dostupnih u Trino-u, moguće je kreirati konektor...“*

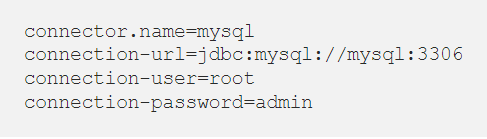
Dakle, ukoliko je moguće podatke iz izvora logički predstaviti u obliku koji *Trino* zahteva, moguće je nad njime kreirati apstrakciju u vidu konektora koji će radnim čvorovima omogućiti da iz izvora pribave podatke. Kreiranje konektora podrazumeva implementiranje *Trino SPI-*ja[[2]](#footnote-2). Kako je *SPI* interfejs, *Trino* može pozivati metode iz interfejsa bez obzira na implementaciju, što implementaciju raspoređivanja i izvršavanja poslova čini nezavisnom od samog izvora podataka.

*Trino* ima određen broj ugrađenih konektora [[4](#TriConcepts)]. Mnoge ekipe programera razvijale su druge konektore po potrebi, nezavisno od *Trino* razvojnog tima, tako da danas postoji veliki broj konektora, pa *Trino* može pristupiti zaista širokom skupu izvora podataka. Spisak i detalji o postojećim konektorima dostupan je u zvaničnoj dokumentaciji [[5](#TriConnectors)].

### Katalozi

*Trino* katalog je komponenta čija je namena da opiše i definiše izvore podataka koje će *Trino* klaster koristiti. Jedan katalog predstavljen je pomoću konfiguracionog fajla koji sadrži važna svojstva kataloga, a kao najvažniji sadrži naziv konektora koji će se koristiti za pristup podacima iz izvora na koji se taj katalog odnosi. Ovaj fajl nosi ekstenziju *.properties*, i zapisi u njemu su oblike *key=value*, gde je *key* naziv svojstva koje se definiše, a *value* njegova vrednost.

Primer jednog konfiguracionog fajla kataloga dat je na slici 2.2. U pitanju je katalog koji opisuje *MySQL* bazu kao izvor podataka. Najvažnija linija u ovom fajlu jeste prva, koja definiše tip konektora, a samim tim i tip izvora podataka, i omogućava *Trino*-u da, pri radu sa ovim izvorom, koristi odgovarajuću implementaciju *SPI*-ja. Ostale linije u fajlu predstavljaju vrednosti koje su karakteristične za sam *MySQL*.



*Slika 2.2 – Primer kataloga za MySQL bazu; preuzeto iz* [[6](#Med)]

Naziv kataloga definisan je onime što prethodi ovoj ekstenziji (npr. ukoliko je fajl sa slike 2.2 nazvan *mysql.properties*, katalog će u *Trino* klasteru nositi naziv *mysql*). Naziv kataloga je važan za jedinstvenu identifikaciju *Trino* tabela. Svaka tabela definisana je katalogom, šemom kojoj pripada i na kraju imenom tabele. Primer iz [[4](#TriConcepts)] objašnjava da je *hive.test\_data.test* jedinstveni naziv tabele *test* iz šeme *test\_data* iz kataloga *hive* (dakle, kataloga definisanog fajlom *hive.properties*). Više o konceptima šeme i tabele sledi u narednim poglavljima.

Kako naziv konfiguracionog fajla može biti proizvoljno izabran i nije ni u kakvoj vezi sa nazivom konektora na koji se katalog odnosi, moguće je kreirati više kataloga za rad sa različitim instancama istog tipa izvora podataka (npr. više kataloga za rad sa različitim instancama *MySQL-*a, *Hive*-a...).

### Šeme

Šema predstavlja skup određenog broja tabela iz konkretnog izvora podataka. Svaka šema mora da pripada katalogu, tako da naziv kataloga i naziv šeme jedinstveno određuju skup tabela. Koncept šeme odgovara konceptu baze u relacionim bazama, a ima pandan i u nekim drugim tehnologijama za čuvanje podataka, kao što je *Hive*. Preslikavanje na *Trino* šemu u ovakvim izvorima podataka trivijalno. Kreiranje šema u izvorima gde taj koncept ne postoji zavisi od implementacije konektora i načina na koji programeri koji konektor razvijaju odluče da organizuju podatke, koji su prethodno već organizovani u tabele.

### Tabele

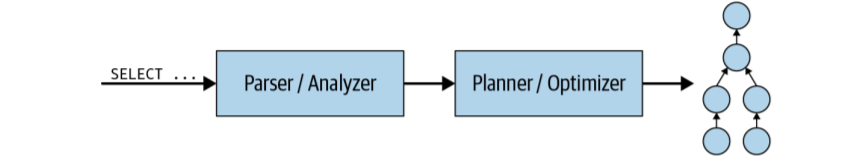
Tabele u *Trino*-u predstavljaju isti koncept kao i tabele u klasičnim relacionim bazama – vrste podeljene na polja definisana kolonama tabele. Kolone i ovde imaju naziv i tip. Kao i kada je šema u pitanju, preslikavanje između tabele iz izvora podataka i tabele iz *Trino*-a zavisi od implementacije konektora, a može biti trivijalno ukoliko su podaci u samom izvoru prirodno organizovani u tabele.

## Model izvršavanja upita

Kao i sa izvorima podataka, *Trino* definiše i nekoliko apstraktnih pojmova koji omogućavaju bolje razumevanje i veću kontrolu nad izvršavanjem upita.

### Plan upita

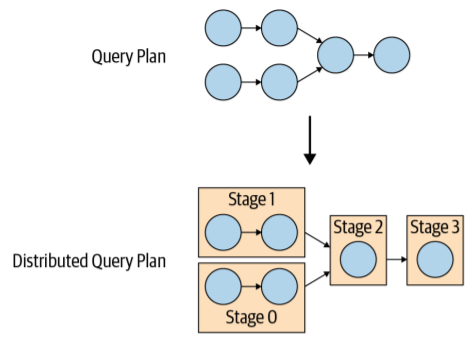
Klijent svoje upite izdaje koordinatorskom čvoru u formi tekstualnog iskaza (razlika između *iskaza* i *upita* opisana je u fusnoti u poglavlju 2.1.1). Koordinator zatim parsira i analizira taj tekst, pa se tako obrađena forma iskaza propušta kroz koordinatorsku komponentu za planiranje i optimizaciju upita (više o detaljima planiranja i optimizacije upita može se naći u *Trino* vodiču [[2](#Ful21)], u poglavlju 4). Kao rezultat svih ovih akcija, dobija se *plan upita*, što je posebna *Trino* struktura koja na visokom nivou, bez previše detalja, određuje niz koraka koje je potrebno izvršiti kako bi se realizovalo ono što je klijent zahtevao svojim *SQL* iskazom. Na slici 2.3 prikazan je proces transformacije iskaza u plan izvršavanja. Struktura nalik na stablo na slici predstavlja plan upita.



*Slika 2.3 – Proces transformacije SQL iskaza u plan izvršavanja; preuzeto iz* [[2](#Ful21)]

### Faze izvršavanja i fragmenti plana

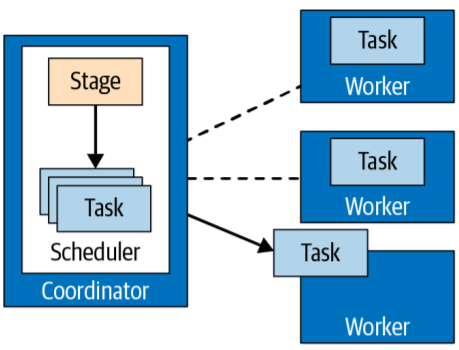
Nakon što je kreiran plan upita kao jedinstvena celina, on se prevodi u distribuiran oblik podelom na *fragmente plana*. Fragment plana u toku izvršavanja naziva se *faza izvršavanja* (eng. *stage*). Podela na fragmente dovodi do nastanka stabla zavisnosti u kojem su čvorovi fragmenti, odnosno faze (slika 2.4). Broj faza zavisi od kompleksnosti početnog iskaza, odnosno upita koji se od njega dobija. Zavisnosti među fazama utiču na mogućnost paralelizacije izvršavanja. Ipak, kako bi se maksimalno iskoristilo postojanje više radnika u klasteru i sprovela maksimalna moguća paralelizacija izvršenja, faze nisu te koje se dodeljuju cele na izvršenje jednom radniku. Naime, faze se dalje dele na nezavisne jedinice izvršenja, odnosno *zadatke*.



*Slika 2.4 – Preslikavanje osnovnog plana upita na faze izvršenja; preuzeto iz* [[2](#Ful21)]

### Zadaci

Zadaci (eng. *task*) su jedinice izvršenja od kojih se sastoje faze. Svaka faza sastoji se od jednog ili više zadataka koji obrađuju deo podataka. Koordinator je zadužen za podelu zadataka iz jedne faze radnim čvorovima, kako je prikazano na slici 2.5. Zadaci se mogu izvršavati u paraleli čime se smanjuje ukupno vreme potrebno za izvršenje jedne faze.

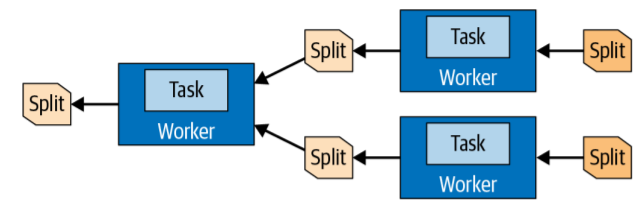


*Slika 2.5 – Podela zadataka jedne faze po radnim čvorovima; preuzeto iz* [[2](#Ful21)]

### *Split*-ovi[[3]](#footnote-3)

*Split* predstavlja onaj deo čitavog skupa podataka koji obrađuje jedan zadatak. To je zapravo podskup podataka za koji je u određenom trenutku zadužen jedan radnik (ali jedan radnik može u trenutku raditi sa više *split*-ova). Izgled *split*-a i način na koji se on kreira od čitavog skupa podataka zavisi od izvora podataka sa kojim se radi, kao i od implementacije konektora. U [[2](#Ful21)] se kao primer navodi *Hive* kod kojeg je *split* definisan putanjom do fajla na koji se odnosi, kao i *offset*-om[[4]](#footnote-4) i dužinom kojima je određeno koji deo fajla se preslikava na dati *split*.

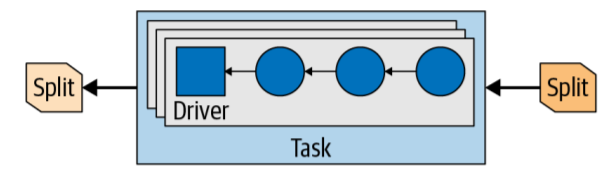
U pogledu toga odakle pribavljaju *split*-ove, postoji razlika između zadataka koji pripadaju fazama koji su listovi u stablu zavisnosti između faza, i onih faza koje se nalaze na višim nivoima. Faze u listovima, tačnije njihovi zadaci, pribavljaju podatke svog *split*-a iz izvora podataka na način koji definiše konektor. Za viši nivo, izvor *split*-a jeste izlaz iz zadataka sa nivoa ispod. Zadaci na najdubljem nivou su oni koji se prvi izvršavaju, pribavljaju podatke u vidu *split*-ova iz izvora, i u svoj izlazni bafer smeštaju nove *split*-ove koji sadrže podatke nastale kao rezultat obrade izvornog *split*-a, a koji se takođe nalaze u tabelarnoj formi. Rezultujući *split*-ovi smeštaju se u izlazni bafer, gde postaju dostupni za ponovnu obradu. Zadaci iz faze na sledećem nivou pribavljaju sada podatke kao *split*-ove iz izlaznih bafera zadataka na prethodnom nivou, i ponovo rezultate obrade smeštaju u bafer. Na kraju, koordinator može preuzeti konačan rezultat iz izlaznih bafera zadataka na najvišem nivou. Na slici 2.6 prikazan je tok podataka kroz zadatke, počev od izvornih *split*-ova. Na desnom kraju prikazane strukture nalaze se *split*-ovi iz originalnog izvora podataka.



*Slika 2.6 – Tok podataka kroz zadatke počev od izvornih split-ova; preuzeto iz* [[2](#Ful21)]

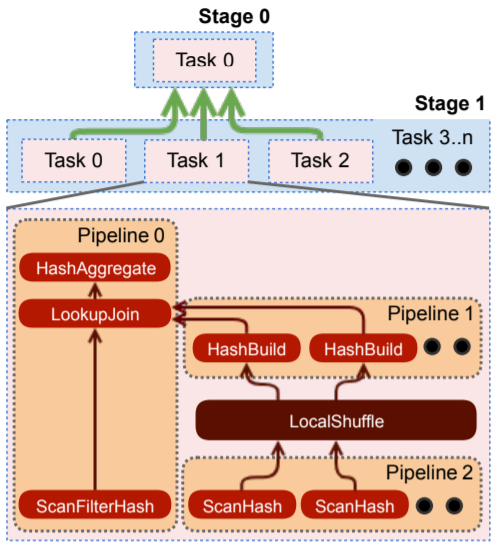
### Drajveri i operatori

Zadaci obrađuju *split*-ove tako što nad njima izvrše operacije definisane **operatorima**, kao što su prolazak kroz tabele radi čitanja, filtriranje, grupisanje, agregacije, spojevi... Dakle, u pitanju su operacije već poznate u *SQL*-u. Operatori se grupišu u sekvence od kojih svaka izvršava neki deo čitavog zadatka. Svaka instanca sekvence operatora predstavlja jedan **drajver** (eng. *driver*). U jednom zadatku koji radi sa jednim *split*-om moguće je imati više sekvenci operatora, odnosno više drajvera, kao što je prikazano na slici 2.7. Stoga, drajver predstavlja jedinicu paralelizma najnižeg nivoa. Zadaci i *split*-ovi definisali su globalni paralelizam, na nivou klastera, u pogledu raspoređivanja posla na čvorove radnike koji ih mogu izvršavati nezavisno, odnosno u paraleli. Drajveri, sa druge strane, predstavljaju osnovnu jedinicu paralelizma lokalnog tipa, na jednom čvoru. Drajver ima jedan ulaz i jedan izlaz. Optimizator izvršenja upita je u stanju da, nakon što je određena sekvenca operatora koja je neophodna za izvršenje zadatka, tu sekvencu ako je moguće podeli na više manjih, odnosno da kreira više drajvera, koji se mogu izvršavati u paraleli.



*Slika 2.7 – Paralelni drajveri u jednom zadatku; preuzeto iz* [[2](#Ful21)]

Ponekad, iako se nakon analize zadatka kreira više drajvera, nije moguće sve njih izvršavati u paraleli. Dobar primer, dat u [[3](#Set19)], jeste heš-spoj[[5]](#footnote-5). Na slici 2.8 prikazana je podela posla na drajvere (na slici označeni sa *pipeline*). Zadatak 1 radi sa dva *split*-a, i mora da pročita podatke iz oba (bilo da dolaze iz osnovnih izvora podataka, ili iz izlaznih bafera zadataka na nižem nivou). Drajver 0 (odnosno *pipeline* 0) predstavlja deo algoritma heš-spoja koji pretražuje heš tabelu (eng. *probe* deo) i radi sa levom stranom spoja, a drajveri 1 i 2 deo za kreiranje heš tabele (eng. *build* deo) u koju se smešta desna strana spoja[[6]](#footnote-6). Kako je u ovom slučaju za *build* fazu neophodno pročitati podatke iz *split*-a i kreirati tabelu, *build* deo je podeljen na dva drajvera, jedan koji učitava podatke (drajver 2) i drugi koji kreira memorijsku strukturu heš tabele (drajver 1). Ova dva drajvera su nezavisna i mogu se izvršavati u paraleli u posebnim nitima, s tim što je posle neophodno podatke iz *split*-a smestiti u tabelu, što je prikazano *LocalShuffle* fazom. Iako je drajver 0 izdvojen kao posebna sekvenca operatora, on za vreme dok se izvršavaju drajveri *build* faze može samo da učita svoj *split* (označeno sa *ScanFilterHash*), pa mora da čeka na kraj kreiranja heš tabele. Svakako, ovo je samo jedan specifičan slučaj, i ne treba na osnovu njega steći utisak da je postojanje drajvera besmisleno, naprotiv, drajveri znatno ubrzavaju i poboljšavaju rad *Trino* klastera.



*Slika 2.8 – Podela izvršavanja heš-spoja na drajvere; preuzeto iz* [[3](#Set19)]

# Praktični primeri primene *Trino*-a

U ovom poglavlju opisan je primer primene *Trino* tehnologije za rad sa dva izvora podataka – *MySQL* relaciona baza podataka i *Apache Hive*. Za realizaciju primera korišćeni su *Docker* i *docker-compose*, ali je moguće *Trino* distribuciju instalirati lokalno, po uputstvu iz dokumentacije [[7](#TriInstall)]. Primer opisan u ovom poglavlju sastavljen je po uzoru na druge primere dostupne na *github* repozitorijumu <https://github.com/bitsondatadev/trino-getting-started> [[8](#GitTrinoGetStarted)]. U ovom repozitorijumu dati su primeri korišćenja *Trino* tehnologije za pojedinačni pristup raznovrsnim izvorima podataka. Ono što je opisano u ovom poglavlju jeste svojevrsno proširenje tih primera koje kreira *Trino* klaster za rad sa dva izvora podataka (*MySQL* i *Hive*).

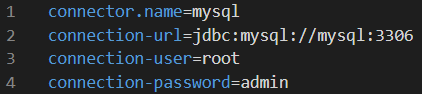
Primer opisan u ovom poglavlju nalazi se u *github* repozitorijumu na linku <https://github.com/petarTrifunovic98/asvsp-seminarski> [[9](#GitExampleRepo)].

## Katalozi

Da bi *Trino* znao sa kojim izvorima podataka radi, neophodno je kreirati odgovarajuće kataloge (detaljnije o katalozima u poglavlju 2.2.2), odnosno napisati njihove konfiguracione fajlove. Po kloniranju repozitorijuma [[9](#GitExampleRepo)], ovi fajlovi se nalaze u direktorijumu na putanji **asvsp-seminarski/Example/compose-mysql-hive/etc/catalog**, ali se u samom *Trino*-u (u njegovom *Docker* kontejneru) nalaze na putanji **etc/trino/catalog**, jer je *catalog* ime direktorijuma u kome *Trino* očekuje da nađe fajlove koji opisuju kataloge, i mora biti ispod *etc* direktorijuma. Prebacivanje ovih fajlova iz lokalnog fajl sistema na fajl sistem *Docker* kontejnera izvršen je korišćenjem *bind-mount*-ova[[7]](#footnote-7).

### Katalog za *MySQL*

Na slici 3.0 prikazan je izgled *MySQL* kataloga, odnosno fajla *mysql.properties*. Fajl, odnosno konfiguracija kataloga, preuzeta je iz [[8](#GitTrinoGetStarted)]. Prva, najvažnija linija, označava koji konektor treba koristiti za pristup podacima u *MySQL*-u. Zatim slede parametri karakeristični za *MySQL*, odnosno adresa i port preko koje treba vršiti konekciju, a zatim slede kredencijali korisnika. Kredencijali su postavljeni putem promenljivih okruženja u *docker-compose.yml* fajlu (poglavlje 3.3.3). Više o konektoru za *MySQL* može se naći u zvaničnoj dokumentaciji za ovaj konektor [[10](#TriMySQLConn)].



*Slika 3.0 – Konfiguracioni fajl za MySQL bazu podataka*

Kako je naziv fajla *mysql.properties*, to će katalog nastao na osnovu ovog fajla nositi naziv *mysql*.

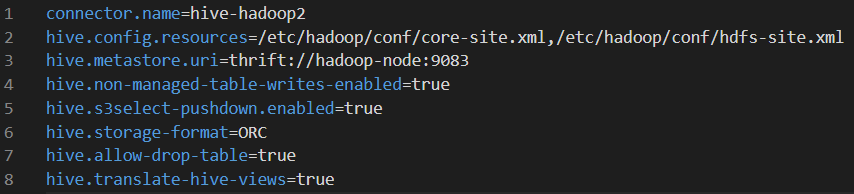
### Katalog za *Apache Hive*

Na slici 3.1 prikazan je izgled *Apache Hive* kataloga, odnosno fajla *hdfs.properties*. Fajl, odnosno konfiguracija kataloga preuzeta je iz [[11](#DocBindMount)]. Prva linija, kao i u slučaju *MySQL*-a, označava koji konektor će biti korišćen za pristup podacima u *Apache Hive*-u. U nastavku su opisane još neke od linija iz konfiguracionog fajla. O ostalim parametrima iz fajla, kao i onima kojih nema u fajlu, više se može naći u zvaničnoj dokumentaciji za *Hive* konektor [[12](#TriHiveConn)].

Druga linija navodi putanju do konfiguracionih fajlova za *HDFS* (nad kojim radi *Hive*). Ovi fajlovi se, po kloniranju repozitorijuma [[9](#GitExampleRepo)], nalaze lokalno u direktorijumu na putanji **asvsp-seminarski/Example/compose-mysql-hive/conf**, ali se, ponovo korišćenjem *bind-mount*-ova, prebacuju u *Trino Docker* kontejner na putanju **etc/hadoop/conf**.

Treća linija navodi adresu i port do *Hive* *metastore*-a[[8]](#footnote-8).

Šesta linija postavlja *ORC* format kao podrazumevani format fajla koji se koristi pri kreiranju novih tabela. *ORC* je format koji smešta podatke po kolonama [[13](#Lev18)].



*Slika 3.1 – Konfiguracion fajl za Apache Hive*

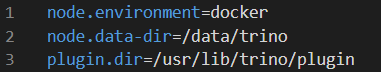
Imajući u vidu naziv fajla, na osnovu njega će se kreirati katalog sa nazivom *hdfs*.

## Konfiguracija instance

Konfiguracija jedne instance definisana je pomoću dva fajla opisana u narednim poglavljima. Opisani primer ima samo jednu instancu koja radi i kao koordinator i kao radnik.

### Konfiguracija čvora

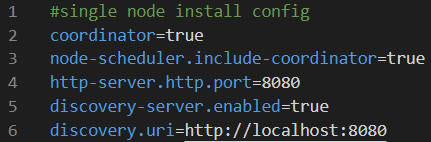
Konfiguracija samog čvora nalazi se u fajlu koji mora da nosi naziv *node.properties*. Po kloniranju repozitorijuma [[9](#GitExampleRepo)], fajl se može naći na putanji **asvsp-seminarski/Example/compose-mysql-hive/etc/node.properties**, što će se preslikati na putanju **etc/trino/node.properties** u *Trino* kontejneru. Izgled ovog fajla nalazi se na slici 3.2. Prve dve linije su od interesa. Prva određuje naziv okruženja u kome se nalazi čvor, i svi čvorovi jednog klastera moraju biti u istom okruženju. Druga linija definiše putanju na kojoj će se (u *Trino* kontejneru) čuvati logovi i ostali važni podaci. Takođe, u ovom fajlu moguće je definisati i polje *node.id* koje mora biti jedinstveno za svaki fajl, odnosno svaki čvor u istom klasteru. Ako se ne navede, vrednost za ovo polje će biti automatski generisano.



*Slika 3.2 – Konfiguracioni fajl čvora*

### Konfiguracija servera

Konfiguracija servera, u pogledu njegove uloge, adrese i sličnih informacija, nalazi se u fajlu sa nazivom *config.properties*. Nakon kloniranja repozitorijuma [[9](#GitExampleRepo)], fajl je na putanji **asvsp-seminarski/Example/compose-mysql-hive/etc/config.properties**, što će se u *Trino* kontejneru preslikati u putanju **etc/trino/node.properties**. Sadržaj fajla nalazi se na slici 3.3.



*Slika 3.3 – Konfiguracioni fajl servera*

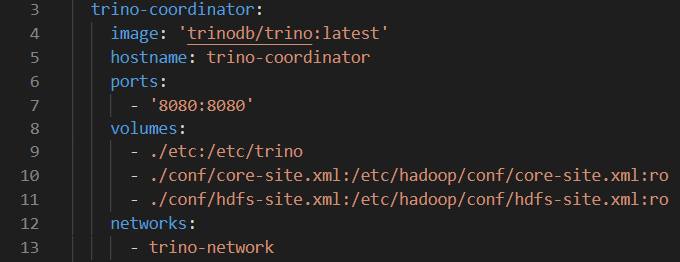
Linija 2 dodeljuje instanci na koju se fajl odnosi ulogu koordinatora. Sledeća linija kaže da bi, pri rasporedli posla (zadataka), trebalo kao radnika uključiti i koordinatora. Linija 4 definiše port na kome je dostupan server. Linija 5 kaže da će servis za otkrivanje čvorova (eng. *discovery service*) biti dostupan na istom portu na kom je dostupan i server na koji se fajl odnosi. Ovaj servis omogućuje radnim čvorovima da se „prijave“ kod koordinatora kako bi on bio u stanju da im dodeli posao. Poslednja linija instanci na koji se fajl odnosi govori koji su adresa i port servisa za otkrivanje (što je u ovom slučaju isto kao i adresa i port instance za koju se piše ovaj fajl).

## *Docker-compose* fajl

U direktorijumu **asvsp-seminarski/Example/compose-mysql-hive** kloniranog repozitorijuma [[9](#GitExampleRepo)] nalazi se fajl *docker-compose.yml* koji definiše sve servise potrebne za izvršavanje primera. Svi servisi komuniciraju preko *trino-network* mreže.

### Servis za *Trino*

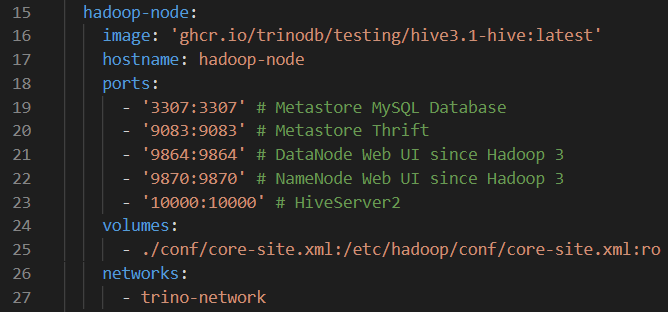
Servis za *Trino* čvor definisan je delom *docker-compose.yml* fajla prikazanim na slici 3.4. U delu *volumes* definisana su pomenuta preslikavanja lokalnih putanja na *bind-mount*-ove.



*Slika 3.4 – Deo docker-compose.yml fajla koji definiše servis za Trino čvor*

### Servis za *Hive*

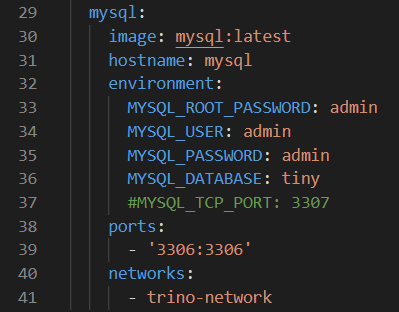
Servis za *Hive* definisan je delom *docker-compose.yml* fajla prikazanim na slici 3.5.



*Slika 3.5 – Deo docker-compose.yml fajla koji definiše servis za Hive*

### Servis za *MySQL*

Servis za *MySQL* definisan je delom *docker-compose.yml* fajla prikazanim na slici 3.6. U *environment* delu definisane su promenljive okruženja, i to šifra za *root* korisnika, korisničko ime i šifra novog korisnika, kao i naziv baze koja će se koristiti.



*Slika 3.6 – Deo docker-compose.yml fajla koji definiše servis za MySQL*

## Pokretanje primera i izvršavanje upita

Primer se pokreće pozicioniranjem u **asvsp-seminarski/Example/compose-mysql-hive** direktorijum kloniranog repozitorijuma [[9](#GitExampleRepo)] i izvršavanjem komande *docker-compose up –d*. Nakon toga, treba pristupiti *Trino* kontejneru, komandom *docker exec –it compose-mysql-hive\_trino-coordinator\_1 trino* komande. Nakon ovoga, pojavljuje se prompt *Trino* *CLI*-a prikazan na slici 3.7.



*Slika 3.7 – Trino CLI prompt*

Sada, ukoliko se izvrši komanda *show catalogs;* dobiće se spisak postojećih kataloga. Između ostalog, među njima su *hdfs* i *mysql*. Takođe, tu se nalazi i *tpch* katalog koji sadrži test podatke, konkretno u šemi po nazivu *tiny*, koji će se u primeru prebaciti u *Hive* i *MySQL*.

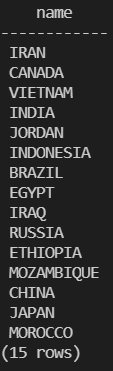
*MySQL* takođe već sadrži *tiny* šemu, odnosno bazu, jer je taj naziv baze definisan putem *MYSQL\_DATABASE* promenljive okruženja u *docker-compose.yml* fajlu. Važno je razgraničiti razliku između koncepta šeme u relacionim bazama, i *Trino* šeme. *Trino* šema je ono što odgovara jednoj konkretnoj bazi u *MySQL*-u, u ovom slučaju po imenu *tiny*. Koncept šeme u relacionim bazama, sa druge strane, definiše unutrašnju strukturu baze u pogledu tabela koje ona sadrži.

*Hive*, odnosno katalog po imenu *hdfs*, ne sadrži *tiny* šemu. Treba je kreirati izvršavanjem komande *CREATE SCHEMA hdfs.tiny;*.

Sada, kada i *mysql* i *hdfs* katalozi imaju *tiny* šemu, treba kreirati tabele. Prvo, na osnovu *tpch.tiny.customer* tabele treba kreirati *mysql.tiny.customer* tabelu, izvršavanjem komande *CREATE TABLE mysql.tiny.customer AS SELECT \* FROM tpch.tiny.customer;*. Dakle, korišćenjem CREATE TABLE AS operacije, svi podaci iz *tpch.tiny.customer* tabele kopiraju se u novu *mysql.tiny.customer* tabelu. Po istom principu kreira se i tabela *hdfs.tiny.nation*  - izvršavanjem komande *CREATE TABLE hdfs.tiny.nation AS SELECT \* FROM tpch.tiny.nation;*.

Sada se može osetiti snaga *Trino*-a i videti kako posmatra na jedinstven način sve izvore podataka sa kojim radi. Nad dve kreirane tabele moguće je izvršavati upite zasebno, ali je takođe moguće kroz jedan upit obratiti se obema tabelama. Dobar primer za to jeste izvršavanje spoja, kroz, recimo, upit sledećeg oblika:

*SELECT DISTINCT h.name   
FROM hdfs.tiny.nation AS h, mysql.tiny.customer AS m   
WHERE h.nationkey=m.nationkey AND m.custkey BETWEEN 1000 AND 1020;*



*Slika 3.8 – Rezultat navedenog upita*

Na slici 3.8 prikazan je rezultat koji vraća upit. Rezultat će biti drugačiji pri svakom pokretanju primera, zbog nasumičnog generisanja *tpch* podataka.

# *Presto* performanse

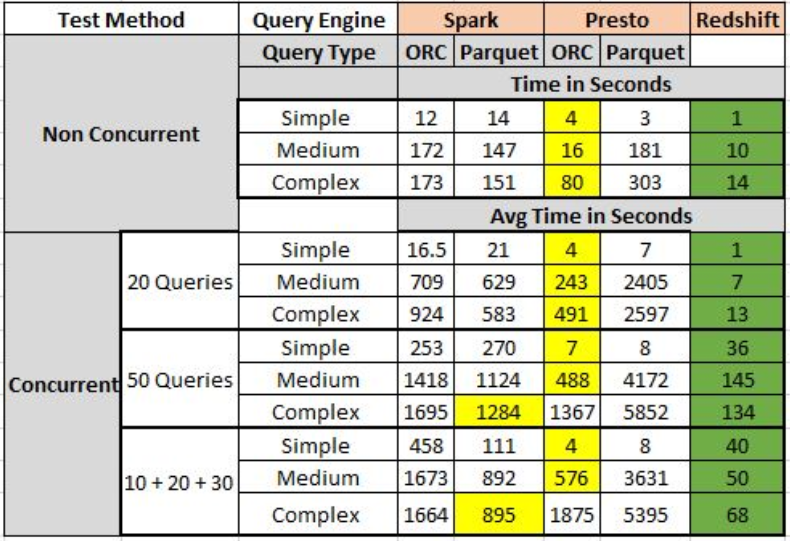
*Trino* (tačnije *Presto*) jeste razvijen kao alternativa *Hive*-u, s obzirom na to da performanse *Hive*-a nisu bile zadovoljavajuće. Ipak, treba napomenuti da ovo nije dovelo do potpunog napuštanja *Hive*-a kao tehnologije, naprotiv, ona je nastavila da bude razvijana dalje tako da danas postoje verzije i optimizacije *Hive*-a koje daju bolje performanse od *Trino*-a. Ovakvo nešto donekle jeste očekivano – *Hive* je razvijen za rad sa distribuiranim fajl sistemima, (*HDFS*, *Amazon S3*...) i način njegovog rada prilagođen je ovakvim izvorima podataka. Sa druge strane, *Trino* mora da održi strukturu opisanu u poglavlju 2 koja mu omogućava rad sa raznovrsnim izvorima podataka. Svakako, velika prednost *Trino*-a u odnosu na *Hive* jeste mogućnost rada nad mnogo širim skupom izvora podataka. Kao što je i inače slučaj, ni za jedno ni za drugo rešenje ne može se reći da je savršeno, i u zavisnosti od slučaja treba odlučiti koje rešenje bi bilo pogodnije.

U nastavku će biti dat pregled performansi tehnologije opisane u ovom radu kroz nekoliko primera rezultata sprovođenja *benchmark* testova. Naziv poglavlja je *Presto performanse* zbog toga što su relevantni testovi na internetu, koji daju objektivan uvid u performanse ove tehnologije, dostupniji za *Presto* nego za *Trino*.

## *Apache Hive, Apache Spark* i *Presto*

Dobar pregled razlike u performansama između *Hive*-a, *Spark*-a i *Presto*-a dat je na [[14](#All19)]. U ovom primeru testiranja kao skladište korišćen je *Amazon S3*. Logički gledano, primer radi sa dve distribuirane tabele, od kojih prva ima približno 6 milijardi redova, a druga oko 589 000 (izvor ne navodi veličinu korišćenih podataka u pogledu bajtova). Za testiranje su korišćena tri upita – prvi je jednostavan *SELECT*; drugi sadrži spojeve i filtriranje; treći sadrži spojeve, filtriranje i agregaciju. Filtriranje kod trećeg upita je blaže nego kod drugog, kako bi se agregacija radila nad većim brojem podataka.

Na slici 4.0 prikazani su rezultati u pogledu prosečnog vremena izvršenja upita. Oznake *simple, medium, complex* su redom oznake za tri upita opisana u pasusu iznad. Informacije za *Hive* nisu date u tabeli jer su, kako je navedeno, izvršavanja bila znatno sporija od *Spark*-a i *Hive*-a, pa nisu svi testovi ni mogli da budu izvršeni do kraja. Verzija *Hive*-a korišćena u ovom primeru jeste *Hive 2.3.0.* Gornji deo tabele (*non concurrent*) daje informacije o brzini izvršavanja upita kada su se izvršavali pojedinačno (po jedan upit u jednom trenutku). Donji deo (*concurrent*) odnosi se na situacije kada je istovremeno zahtevano izvršavanje više upita, i to 20, 50, pa na kraju po 10, 20 i 30 zadavanja upita u trenutku, sa kratkom pauzu između svakog skupa od više upita. Dodatno, ubačeni su i podaci o izvršavanju pomoću *Redshift*-a, koji nisu preterno relevantni za razmatranje u smislu upoređivanja, s obzirom na to da su njegovi rezultati višestruko bolji. Tabela takođe daje uvid u performanse u zavisnosti od formata fajla koji se koristi za čuvanje podataka. Oba formata (i *Parquet* i *ORC*) smeštaju podatke po kolonama.



*Slika 4.0 – Rezultati izvršavanja upita pomoću Sparka-a, Presto-a i Redshift-a; preuzeto iz* [[14](#All19)]

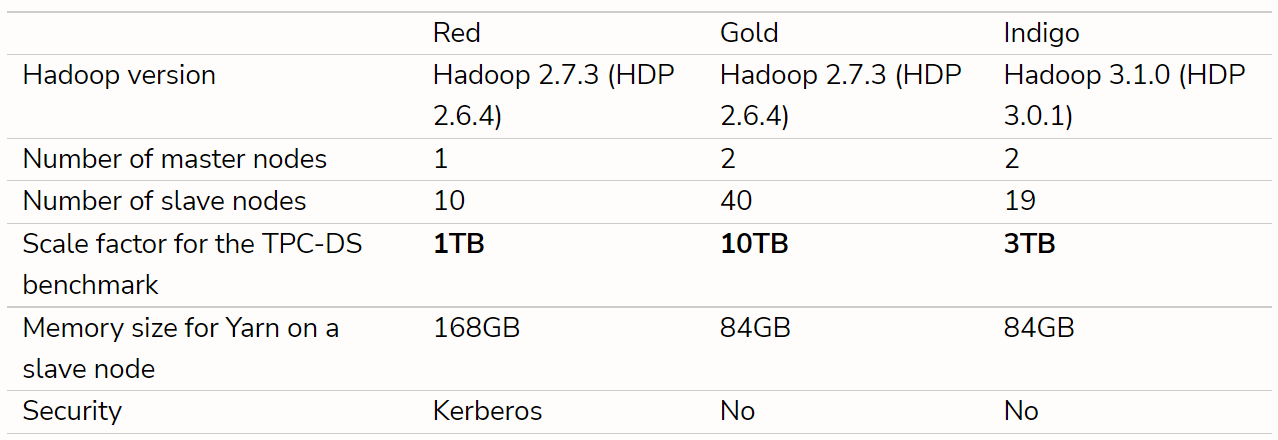
Ono što se može videti iz tabele jeste da je *Presto* brži od *Spark-*a kada su u pitanju nekonkurentna izvršavanja. Kada se više upita izvršava istovremeno, *Spark* se pokazao bolje kada je konkurentnost velika i kada je upit kompleksan. *Presto* i dalje pokazuje znatno bolje vreme izvršavanja kada je je konkurentnost kao i kompleksnost upita umerena. Takođe, iz tabele se vidi da je *ORC* format fajla znatno bolji izbor za *Presto* od *Parquet* formata.

Na *web* stranici [[14](#All19)] moguće je naći još nekoliko interesantnih dijagrama koji pružaju drugačiji vid vizuelizacije podataka sažetih u tabeli na slici 4.0.

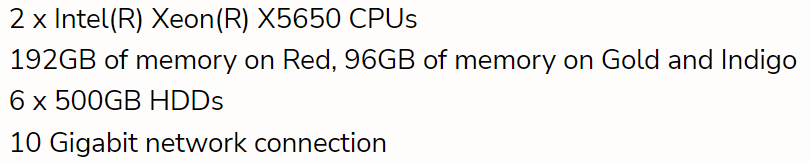
## *Apache Spark*, *Presto* i različite verzije *Apache Hive*-a

Drugi izvor [[15](#Dat18)] upoređuje takođe *Spark, Presto* i *Hive*, s tim što pruža uvid u razliku koja nastaje u zavisnosti od verzije *Hive*-a koja se koristi i pokazuje kako novije verzije *Hive*-a mogu da daju bolje performanse od *Presto*-a.

Testiranje performansi izvršeno je korišćenjem tri različite konfiguracije klastera (*Red, Gold, Indigo*). U sva tri slučaja korišćen je *TPC-DS benchmark* [[16](#TPC)]*.* Kao izvor podataka korišćen je *HDFS*. Tri pomenute konfiguracije klastera date su na slici 4.1. Slika 4.2 pruža informacije o korišćenim mašinama.

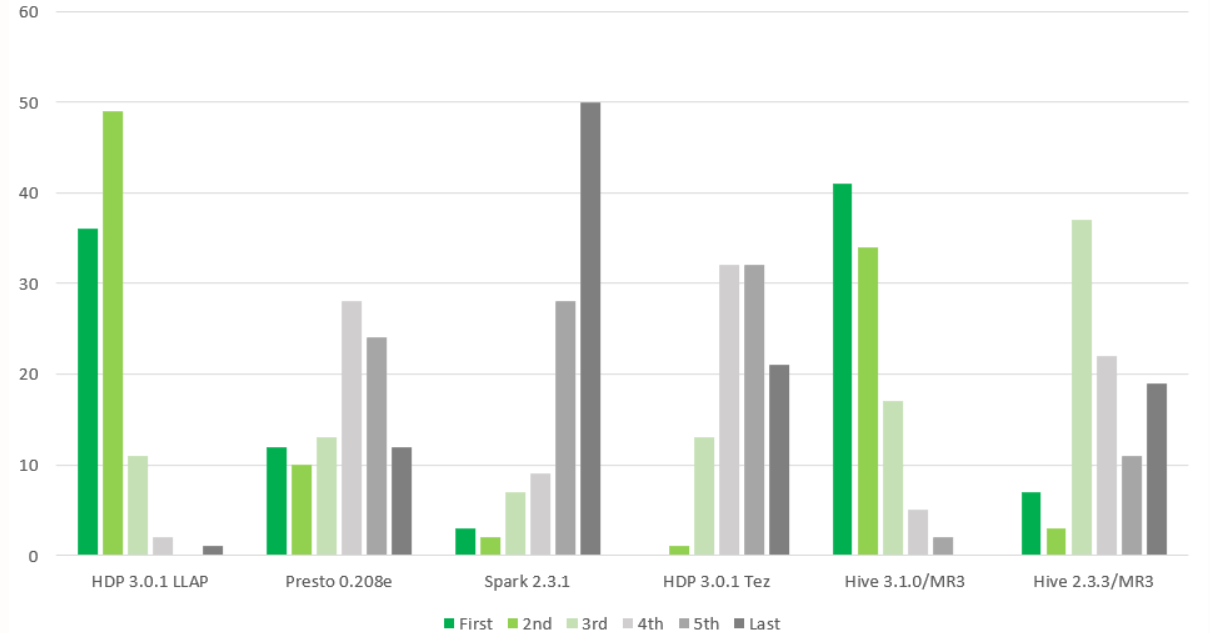


*Slika 4.1 – Tri različite konfiguracije klastera; preuzeto iz* [[15](#Dat18)]



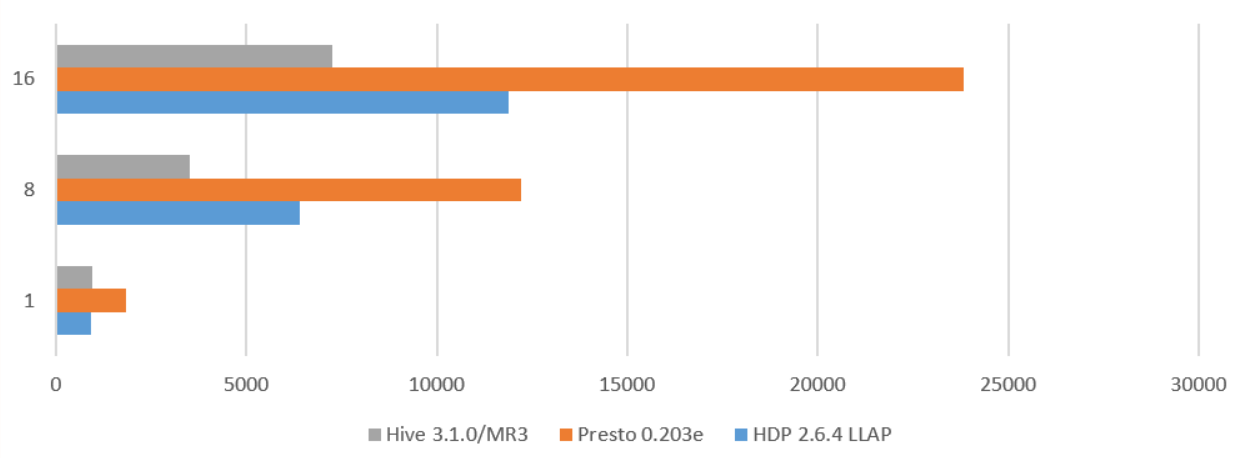
*Slika 4.2 – Osobine korišćenih mašina; preuzeto iz* [[15](#Dat18)]

U ovom poglavlju biće opisani rezultati za *Indigo* klaster pri zadavanju samo jednog upita u trenutku (slika 4.3), i za *Gold* klaster kada se izdaje više upita istovremeno (slika 4.4). Rezultati za sve slučajeve (sva tri klastera i za pojedinačne i za višestruke upite) mogu se naći na [[15](#Dat18)].



*Slika 4.3 – Rezultati Indigo klastera pri zadavanju pojedinačnih upita; preuzeto iz* [[15](#Dat18)]

Rezultati na slici 4.3 pokazuju koliko puta (za koji broj upita iz *TPC-DS benchmark-*a) je koja od tehnologija bila najbrža (*first*), koliko puta druga po redu (*2nd*) i tako dalje. Može se videti da su *LLAP* verzija *Hive*-a, kao i verzija 3.1.0 najviše puta bile na prvom ili drugom mestu po brzini izvršavanja. *Presto* je najviše puta bio na četvrtom ili petom mestu, dok se *Spark* za najveći broj upita pokazao najgore. Ako posmatramo samo *Presto, Spark* i 2.3.3 verziju *Hive*-a, rezultati donekle liče na situaciju opisanu u prethodnom poglavlju, gde je *Presto* najčešće davao najbolje rezultate.



*Slika 4.4 – Rezultati Gold klastera pri zadavanju od 1 do 16 istovremenih upita*

Kada su u pitanju konkurentna izvršavanja, na slici 4.4 može se videti prosečno vreme izvršavanja (horizontalna osa) u zavisnosti od broja istovremenih upita (vertikalna osa). Može se videti da *Presto* daje znatno lošije rezultate od obe prikazane verzije *Hive*-a.

# Zaključak

*SQL* je pre pojave sistema za rad sa velikim skupovima podataka bio široko korišćen i vrlo dobro prihvaćen, kako među ljudima koji se bave računarstvom i informacionim tehnologijama, tako i u mnogim drugim oblastima (ekonomija, finansije, bankarstvo...). Može se reći da je *SQL* postao standard, i svakako predstavlja izuzetno pogodan način za izvršavanje upita i pribavljanje podataka iz relacionih baza. Sa druge strane, relacione baze nisu pogodno rešenje za razvoj danas sve češćih sistema za rad sa velikim skupovima podataka. *Trino* uspeva da *SQL* prilagodi ovakvim sistemima i da pristup i vršenje upita nad raznovrsnim izvorima podataka svede na oblike već poznate u relacionim bazama. Posledica ovoga jeste i olakšan pristup podacima na način nezavisan od konkretnog izvora podataka koji se nalazi u pozadini.

# Literatura

x

|  |  |
| --- | --- |
| [1] | Trino. [Online]. <https://trinodb.github.io/docs.trino.io/current/> |
| [2] | M. Fuller, M. Traverso, and M. Moser, *Trino: The Definitive Guide*.: O’Reilly Media, 2021. |
| [3] | R. Sethi et al., "Presto: SQL on Everything," *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019. |
| [4] | Trino. [Online]. <https://trino.io/docs/current/overview/concepts.html> |
| [5] | Trino. [Online]. <https://trino.io/docs/current/connector.html> |
| [6] | B. Olsen. (2021, December) Medium. [Online]. <https://medium.com/geekculture/intro-to-trino-for-the-trinewbie-a5a1088d3114> |
| [7] | Trino. [Online]. <https://trino.io/docs/current/installation.html> |
| [8] | GitHub. [Online]. <https://github.com/bitsondatadev/trino-getting-started> |
| [9] | GitHub. [Online]. <https://github.com/petarTrifunovic98/asvsp-seminarski> |
| [10] | Trino. [Online]. <https://trino.io/docs/current/connector/mysql.html> |
| [11] | Docker Documentation. [Online]. <https://docs.docker.com/storage/bind-mounts/> |
| [12] | Trino. [Online]. <https://trino.io/docs/current/connector/hive.html> |
| [13] | L. Leverenz. (2018, June) Cwiki.apache.org. [Online]. <https://cwiki.apache.org/confluence/display/hive/languagemanual+orc> |
| [14] | (2019, July) All Big Data Things. [Online]. <https://allbigdatathings.blogspot.com/2019/07/hive-vs-spark-vs-presto-sql-performance.html> |
| [15] | (2018, October) Data Monad. [Online]. <https://www.datamonad.com/post/2018-10-30-performance-evaluation-0.4/> |
| [16] | TPC. [Online]. <http://www.tpc.org/tpcds/> |
| [17] | Wikipedia. [Online]. <https://en.wikipedia.org/wiki/Hash_join> |
| [18] | E. Orr. (2021, August) Lakefs.io. [Online]. <https://lakefs.io/hive-metastore-why-its-still-here-and-what-can-replace-it/> |

x

1. U *Trino* dokumentaciji [[1](#Tri)] pravi se jazna razlika između iskaza (eng. *statement*), što jeste tekst zapisan u *SQL-*u i upita (eng. *query*) predstavljenog planom izvršavanja (eng. *query plan*). Kako se u pozadini mogu naći različiti izvori podataka, razlika između ova dva pojma je mnogo veća u *Trino*-u nego u relacionim bazama podataka. [↑](#footnote-ref-1)
2. *Trino Service Provider Interface*; implementiranjem metoda iz ovog interfejsa tako da odgovaraju konkretnom izvoru podataka dobija se *API* putem koga *Trino* može komunicirati sa izvorom. [↑](#footnote-ref-2)
3. U nedostatku adekvatnog prevoda na srpski jezik, zadržan je engleski naziv. [↑](#footnote-ref-3)
4. Udaljenost od početka fajla. [↑](#footnote-ref-4)
5. Eng. *hash join*; više o ovome na [[17](#Wik)]. [↑](#footnote-ref-5)
6. Strane spoja u konkretnom primeru izabrane su tako da odgovaraju slici, tako da levi deo prikazanog zadatka radi sa levom, a desni sa desnom stranom. U realnosti, za *build* stranu heš-spoja najčešće se uzima ona strana spoja koja ima manje redova. [↑](#footnote-ref-6)
7. Više o *bind-mount*-ovima na [[11](#DocBindMount)]. [↑](#footnote-ref-7)
8. Više o *Hive metastore*-u na [[18](#Orr21)]. [↑](#footnote-ref-8)