LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

ORACLE RISINĀJUMU PIELIETOJUMS BIG DATA KONTEKSTĀ

MAĢISTRA DARBS

Autors: **Kristīne Višņevska**

Stud.apl.Nr. kv14013

Darba vadītāja: dr.sc.comp, profesore Laila Niedrīte

RĪGA 2016

ANOTĀCIJA

Maģistra darbā tika apskatīti un analizēti *Oracle* un citu ražotāju piedāvātie risinājumi *Big Data* apstrādei un analīzei. Darbs satur teorētisku dažādu ražotāju rīku apskati un salīdzinājumu, kā arī praktisko daļu, kurā tiek analizētas datu savienošanas iespējas hibrīdā datu noliktavā. Par hibrīdo datu noliktavu tiek uzskatīta tāda datu noliktava, kurā daļa datu atrodas *RDBMS* sistēmā un daļa datu *HDFS* sistēmā.

Darba galvenie mērķi ir analizēt šāda veida sistēmas veiktspēju, izmantojot dažādus datu apvienošanas veidus. Testu laikā tika apskatīti apskatīti gadījumi, kad datu apvienošana notiek *RDBMS* sistēmas pusē un gadījumi, kad datu apvienošana notiek *HDFS* sistēmas pusē, kā arī tika izpētīta un analizēta *Oracle Big Data SQL, Oracle SQL* konektora priekš *HDFS* un *Hive* rīku veiktspēja atkarībā no datu selektivitātes un datu agregācijas izmantošanas.

Atslēgvārdi: Big Data risinājumi, Oracle Big Data rīku analīze, Hibrīdās datu noliktavas modelis, datu apvienošanas veidi, HDFS un RDBMS veiktspēja

ABSTRACT

Thesis: Usage of Oracle solutions in the Big Data context

Master thesis contains overview and analysis of Oracle and other vendor solutions for Big Data processing and analysing. Framework consists of theoretical part where different vendors tools have been compared and described, and experimental part where have been analysed solutions for data joining in hybrid data warehouse. Hybrid data warehouse is warehouse where part of the enterprise data is stored in RDBMS system and part of the date is stored in HDFS system.

Main goals of this thesis is to analyse hybrid system performance by testing different data join types. Tests contained cases when data joining process has been performed on RDBMS side and case when data joining process has been performed on HDFS side. Dependence on data selectivity and aggregation usage has been analysed for Oracle Big Data SQL, Oracle SQL connector for HDFS and Hive tools.

Keywords: Big Data solutions, Oracle Big Data tools analysis, Hybrid Data Warehouse model, data join types, HDFS and RDBMS performance

AUTORREFERĀTS

Darba izstrādes laikā tika veikts gan teorētisks pētījums, kurā tika apskatīti un salīdzināti vairāku ražotāju risinājumi *Big Data* apstrādē, kā arī aplūkota *Hadoop* arhitektūras iespējas un darbības principi.

Praktiskajā daļā tika testētas datu apvienošanas veidi hibrīdajā datu noliktavā, lai veiktu šādus testus tika uzinstalēta eksperimentālā sistēma, kas apvienoja sevī *RDBMS* sistēmu un *HDFS*, kā *RDBMS* datubāze tika izmantota *Oracle* datubāze. Tika veikti datu apvienošanas testi, izmantojot autora izdomātos *SQL* un *Hive QL* pieprasījumus, kas ļāva apskatīt dažādus datu apvienošanas gadījumus. Katrs pieprasījums tika testēts uz trim rīkiem:

- *Oracle Big Data SQL* – veic datu apvienošanu *RDBMS* pusē, darbības laikā izmanto Blūma filtrus un veido *Oracle* ārējās tabulas pa tiešo *HDFS* sistēmā;
- *Oracle SQL* konektors priekš *HDFS* – veic datu apvienošanu *RDBMS* pusē, darbības laikā neizmanto Blūma filtrus un veido ārējās tabulas *Oracle* serverī;
- *Apache Hive* rīks – veic datu apvienošanu *HDFS* pusē, izmanto broadcast datu apvienošanas veidu, darbības laikā nelieto Blūma filtrus.

Testa pieprasījumi tika veidoti tā, lai tiek pārbaudīti dažādi gadījumi – augsta un zema datu selektivitāte, agregācijas veikšanas nepieciešamība. Testu laikā iegūtie rezultāti ir salīdzināmi ar citos pētījumos iegūtajiem rezultātiem

Darbs tika veikts patstāvīgi, ar atsaucēm ir atzīmētas darba daļas un attēli, kur informācija tika ņemta no literatūras avotiem.

SATURS

APZĪMĒJUMU SARAKSTS

API – Application Programming Interface - Lietojumprogrammas saskarne

BI – Business Intelligence - Biznesa inteliģence

CDH - Cloudera Distribution Including Apache Hadoop - Cloudera platforma, kas iekļauj sevī Apache Hadoop

CLI - Command Line Interface - Komandrindas saskarne

CPU – Central Processing Unit - Centrālais procesors

CSV – Comma Separated Values - Komatu atdalītās vērtības

DAG - Directed Acyclic Graph - Vērstais acikliskais grafs

DSS - Decision Support Systems - Lēmumu pieņemšanas sistēmas

ETL – Extract-Transform-Load - Datu izvilkšana, transformēšana un ielāde

ER - Entity–Relationship model - Entītiju-relāciju modelis

GPS - Global Positioning System - Globālā pozicionēšanas sistēma

ID - Identificator - Identifikators

I/O – Input/Output - Ievade/Izvade

IP – Internet Protocol - Interneta protokols

HDFS – Hadoop Distributed File System - Hadoop sadalītā failu sistēma

HDP - Hortonworks Data Platform - Hortonworks platforma

HQL - Hive Query Language - Hive pieprasījumu valoda

HTML - HyperText Markup Language - Hiperteksta iezīmēšanas valoda

JAR - Java Archive - Java arhīvs

JDBC - Java Database Connectivity - Java draiveris savienojumam ar datubāzi

JDK - Java Development Kit - Java izstrādes rīks

JMS - Java Message Service –Java ziņojumu pakalpojums

JSON - JavaScript Object Notation - JavaScript objektu notācijas valoda

JVM – Java Virtual Machine - Java virtuālā mašīna

LDW - Logical Data Warehouse - Loģiskā datu noliktava

NoSQL – Not only Structured Query Language - Ne tikai strukturētā pieprasījumu valoda

ODBC - Open Database Connectivity - Draiveris, kas ļauj pievienoties datubāzei

OLAP - Online Analytical Processing - Tiešsaites datu analītiskā apstrāde

OLTP - Online Transaction Processing - Tiešsaites transakciju apstrāde

OLH - Oracle Loader for Hadoop - Oracle datu ielādētājs priekš Hadoop

PDF - Portable Document Format - Portatīvā dokumenta formāts

PL/SQL – Procedural Language/ Structured Query Language - Procedūru valoda/ Strukturētā pieprasījumu valoda

PMML - Predictive Model Markup Language - Prediktīvā modeļa iezīmēšanas valoda

RAC - Real Application Cluster - Oracle datubāzes klasteris

RAM - Random Access Memory - Operatīvā atmiņa

RDBMS - Relational Database Management System - Relāciju datubāzes pārvaldības sistēma

REST - Representational State Transfer - Reprezentatīvo stāvokļu pārsūtīšana

RTF - Rich Text Format - Teksta failu formāts

SQL – Structured Query Language - Strukturētā pieprasījumu valoda

URI - Uniform Resource Identifier - Universālais Resursu Identifikators

VM - Virtual Machine - Virtuālā mašīna

VPN - Virtual Private Network - Virtuālais privātais tīkls

W3C – World Wide Web Consortium - Vispasaules Tīmekļa konsorcijs

XML - Extensible Markup Language - Paplašināmā iezīmēšanas valoda

IEVADS

Tehnoloģijām attīstoties pieaug arī datu apjoms, kas tiek radīts. Dati var būt gan cilvēku, gan mašīnu radīti, piemēram, dažādi sensori nepārtraukti rada datus par sistēmu stāvokli. Ja agrāk nācās saskarties ar datu glabāšanas problēmu, tad tagad, kad diska vieta ir relatīvi lēta, parādījās cita problēma – šo datu apstrāde. Dati paši par sevi nav informatīvi, lai no tiem iegūtu nepieciešamo un uzticamu informāciju tie ir jāapstrādā.

Darbā tiks apskatītas šobrīd pieejamās lielo datu apstrādes tehnoloģijas, īpaši koncentrējoties uz *Hadoop* un *MapReduce* tehnoloģijām, to darbības principiem, arhitektūru, priekšrocībām un ierobežojumiem. Kā arī *Oracle* un citu ražotāju piedāvātiem risinājumiem *Big Data* apstrādē. Lielākā daļa no šiem risinājumiem ir balstīta uz *Hadoop* sistēmu integrāciju ar *Oracle* relāciju datubāzi. Pamatā uzņēmumi savās transakciju informācijas sistēmās izmanto relācijas datubāzes, kas laika gaitā ir pierādījušas sevi kā uzticamas sistēmas, taču lielo datu gadījumā, to veiktspēja ir zemāka nekā *NoSQ*L datubāzēm un *Hadoop* sistēmā. Tāpēc tiek veidoti risinājumi kā šādas sistēmas integrēt, kā arī nodrošināt reāla laika datu atjaunošanu, jo lēmumi, kas balstās uz novecojušu informāciju, var nebūt optimāli. Tiks apskatītas arī šo rīku citas īpašības, kā piemēram, spēja vizualizēt un apstrādāt lielos datus, sistēmu arhitektūras, kas nodrošina šādu datu apstrādi.

Pirmajā darba daļā tiek apskatīta *Hadoop* tehnoloģija, tās darbības principi un arhitektūra, kā arī detalizēti apskatīts *MapReduce* un *HDFS* darbības principi. Uz *Hadoop* risinājumiem pašlaik tiek balstīti *Big Data* apstrādes un analīzes rīki, jo šī sistēma ir pierādījusi savu efektivitāti.

Otrajā darba nodaļā tiek aplūkoti *Oracle* piedāvātie risinājumi priekš *Big Data*. *Oracle* produkti var tikt klasificēti kā:

- Risinājumi integrācijai ar citiem *Big Data* produktiem un datu glabātuvēm. Šādi produkti ir *Oracle GoldenGate* priekš *Big Data, Oracle Big Data SQL* un *Oracle* konektori priekš dažādām *Hadoop* daļām;
- Risinājumi datu analīzei un vizualizācijai, kā piemēram, *Oracle Big Data Discovery*.

Trešajā nodaļā ir aplūkoti citu lielāko *Big Data* risinājumu ražotāju piedāvātie risinājumi darbā ar *Big Data*. Par pamatu citu ražotāju izvēlei ir kļuvis *Gartner* 2016.gada pētījums par Datu noliktavu un Datu pārvaldes un analīzes risinājumiem [11]. Pēc šī

pētījuma datiem tirgus līderi ir *Oracle*, *Teradata*, *SAP*, *Microsoft* un *IBM*. Nodaļas beigās tiek veikts arī piedāvāto produktu salīdzinājums un analīze, kā arī analizēti citi mēģinājumi realizēt *MapReduce* integrāciju ar *RDBMS*, kā piemēram, izveidot *MapReduce* funkcionalitāti *Oracle* relāciju datubāzē.

Ceturtajā nodaļā tiek aprakstīta eksperimenta implementācija un testēšanas rezultāti. Tiek pētītas datu apvienošanas iespējas hibrīdajā datu noliktavā, kur daļa datu atrodas *HDFS* sistēmā un daļa datu *RDBMS* sistēmā. Datu apvienošanas ātrdarbība tika testēta izmantojot autora izdomātos *SQL* un *Hive QL* pieprasījumus, kas tika izveidoti tādā veidā, lai tiek notestēti gadījumi, kad datu apvienošana notiek pie dažas datu selektivitātes, kā arī testē agregācijas ietekmi uz veiktspēju. Tika testēti trīs rīki – *Oracle Big Data SQL*, *Oracle SQL* konektors priekš *HDFS* un *Apache Hiv*e.

Darba mērķis ir izpētīt un analizēt *Oracle* piedāvātos risinājumus *Big Data* apstrādei un analizēt to veikstspēju. Mērķa sasniegšanai tika izvirzīti uzdevumi:

1. Apskatīt *Big Data* konceptus un īpašības, kā arī izpētīt *Oracle* risinājumus *Big Data* apstrādei;
2. Apskatīt un salīdzināt citu ražotāju piedāvātos risinājumus *Big Data* apstrādei;
3. Testēt *Oracle* piedāvāto rīku veiktspēju, izmantojot dažādus *RDBMS* un *HDFS* datu apvienošanas veidus, salīdzināt rezultātus ar *Apache Hive* rīka rezultātiem;
4. Analizēt testu laikā iegūtos rezultātus un salīdzināt ar citos pētījumos iegūtajiem datiem.

1. HADOOP TEHNOLOĢIJA

Apstrādājot lielos datus nākas saskarties ar vairākām problēmām, kuras tiek klasificētas kā trīs dimensiju problēmas – viena no dimensijām ir datu apjoms, kas tiek radīts un ar to saistītās apstrādes problēmas. Otrā dimensija - datu formātu dažādība un to apstrādes problēmas. Un trešā dimensija – datu radīšanas ātrums un apstrādes ātruma saistītās problēmas. Lielu datu apjoma apstrādes problēma nav jauna un tā tiek risināta jau dažus gadu desmitus, jo datu apjoms nemitīgi pieaug. Vienkāršota pieeja, kas tika pielietota liela apjoma datu apstrādei ir sadalītu datu apstrādes sistēmu arhitektūru radīšana un dažādu programmēšanas valodu izmatošana, lai koordinētu procesus šādā sistēmā. Šī pieeja ļauj atrisināt problēmas, kas ir saistītas ar datu apjoma problēmām, taču netiek galā ar datu radīšanas ātrumu un dažādiem datu formātiem.

Mūsdienās ir izveidojušies vairākas lielo datu apstrādes platformas, kā piemēram, *Hadoop, Hive, Hbase, Cassandra, NoSQL* platformas, tādas kā *MongoDB, Neo4J, Riak* un daudzas citas. Šajā darbā tiks detalizēti apskatīta *Hadoop* un *MapReduce* tehnoloģija un kā tā tiek izmantota *Oracle* piedāvātajos risinājumos lielo datu apstrādē un analīzē. [1,2]

Hadoop ir atvērtā koda programmatūras produkts, kura mērķis ir sadalītā datu glabāšana un sadalītā datu apstrāde, tā tika radīta lielu datu apjomu apstrādei. Visi moduļi *Hadoop* programmatūrā tika izstrādāti pieņemot, ka aparatūras bojājumi ir bieži, tāpēc to radītās sekas tiek automātiski novērstas un sistēma turpina darboties. Pamatā *Hadoop* tehnoloģija sastāv no *HDFS – Hadoop* sadalītā failu sistēmas (*Hadoop Distributed File System*) un *MapReduce* datu apstrādes tehnoloģijas. *Hadoop* sadala failus lielos blokos un tad izsūta šos blokus dažādos klastera mezglos. [2, 3, 29]

1.1. HDFS – Hadoop sadalītā failu sistēma

HDFS ir izveidota kā kļūdu toleranta, mērojama, sadalīta failu sistēma, ko var lietot uz dažāda tipa aparatūras. Tās arhitektūra tika izveidota, lai atrisinātu divas problēmas, ar kurām saskārās izstrādātāji liela apjoma datu apstrādē. Pirmā problēma bija spēja sadalīt failus starp dažādām sistēmām, apstrādāt katru faila daļu neatkarīgi no citām daļām un apstrādes rezultātus apvienot vienotā izvadē. Otrā problēma, ar ko bieži saskārās bija kļūdu tolerance gan failu apstrādes līmenī, gan sadalītās datu apstrādes sistēmā kopumā.

Daži *HDFS* dizaina pieņēmumi:

- Redundance – aparatūrai ir nosliece uz kļūdām, kā arī procesi var pieprasīt vairāk resursus nekā infrastruktūrā ir pieejams, tāpēc ir jāparedz sistēmas redundance, kas varētu palīdzēt atrisināt šādus gadījumus;
- Mērogojamība – ir nepieciešama lineāra mērogojamība datu glabāšanas līmenī, lai varētu nodrošināt paralēlo apstrādi optimālajā līmenī. *HDFS* ir veidots, lai nodrošinātu pilnīgu lineāru mērogojamību;
- Kļūdu tolerance – sistēmas spēja automātiski atjaunoties pēc kļūdām un pabeigt iesākto datu apstrādi;
- Savietojamība starp dažādām platformām – spēja integrēt ar dažādu arhitektūru platformām;
- Apstrāde un datu glabāšana vienotā vidē – dati un to apstrāde notiek vienotā arhitektūrā, tādējādi izvairoties no *I/O* redundances un daudzām piekļuvēm pie disku masīva.

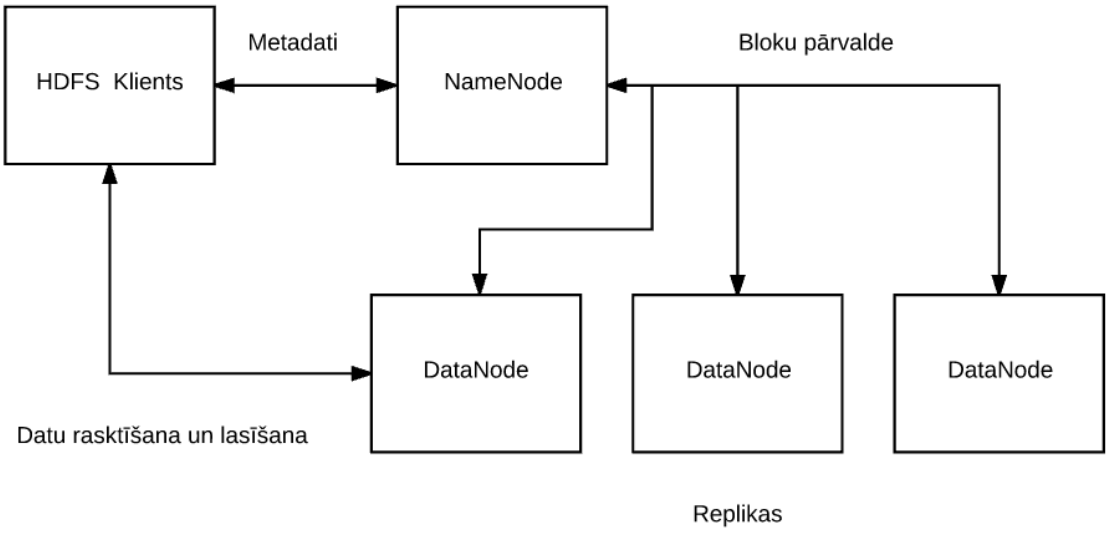
HDFS arhitektūras galvenie mērķi ir:

1. Apstrādāt ļoti liela izmēra failus, mērogā no gigabaitiem līdz petabaitiem;
2. Datu apstrāde reālajā laikā, augsta sistēmas caurlaides spēja;
3. Zemas prasības aparatūras raksturlielumiem – spēja strādāt uz gandrīz jebkuras aparatūras.

Attēlā 1.1 ir parādīta *HDFS* pamata arhitektūra. Tajā ietilps galvenais mezgls – *NameNode*, kas pārvalda visus sistēmas metadatus, mezglu nosaukumus, regulē lietotāju piekļuvi failiem. Kā arī vada visas operācijas ar failiem, tādas kā atvēršana, aizvēršana, pārvietošana, nosaukumu viedošana vai nosaukumu mainīšana failiem vai direktorijām. Galvenais mezgls pārvalda arī visu pakļauto mezglu bloku atrašanās vietas.

Pakļautie mezgli – *DataNodes HDFS* arhitektūrā pārvalda datus un nodrošina šo datu glabāšanu, parasti *HDFS* klasterī ir vairāki tūkstoši pakļauto mezglu un vairāki desmiti tūkstoši lietotāji katrā klasterī, tāpēc *DataNodes* var izpildīt vairākus uzdevumus vienlaicīgi. Pakļautie mezgli ir atbildīgi par rakstīšanas un lasīšanas pieprasījumu pārvaldīšanu, kas ienāk no sistēmas lietotājiem, kā arī bloku uzturēšanu un atbilstošu replicēšanu, lai nodrošinātu redundanci. Bloku pārvalde *HDFS* sistēmā atšķiras no citām failu sistēmām – datu lielums ir vienāds ar bloka garumu. Tas nozīmē, ja bloks ir uz pusi

piepildīts, tad tam ir nepieciešamā vieta uz diska arī ir tikai puse no bloka lieluma. Tas palīdz optimizēt datu glabāšanas kompaktumu un netiek izšķiesta diska vieta, glabājot nepilnos blokus, kā tas ir citās failu sistēmās. [2]



1.1.att. ***HDFS*** **arhitektūra**

Lai nodrošinātu failu sistēmas konsistenci, atjaunojamību un vairāku lietotāju vienlaicīgu darbību arhitektūrā ir iekļauti kontrolpunkta, žurnāla un attēla faili.

Attēla fails glabā metadatus par visu mezglu un bloku nosaukumiem un atrašanās vietām. Kad sistēma tiek startēta galvenais mezgls izdala speciālu atmiņas apgabalu šī faila glabāšanai un atjaunošanai, tā lai vairāki lietotāji vienlaicīgi var tam piekļūt un atrast nepieciešamo informāciju.

Žurnāla failā tiek raksītas visas izmaiņas, kas noteik sistēmā, katra transakcija tiek ierakstīta žurnāla failā. Pirms lietotājs saņem atbildi par savas darbības izpildi tā tiek ierakstīta un sinhronizēta ar žurnāla failu, tādējādi startēšanas brīdī galvenais mezgls var izmantot šo failu, lai atjaunotu sistēmu tajā stāvoklī, kādā tā bija pirms izslēgšanas.

Kontrolpunkts arī ir izveidots, lai varētu veikt sistēmas atjaunošanos. Pastāvīgs ieraksts lokālajā failu sistēmā par *HDFS* sistēmas stāvokli noteiktajā laika punktā tiek uzskatīts par kontrolpunktu. Pēc sistēmas uzstartēšanās tas nekad netiek modificēts vai atjaunots. Nākamais kontrolpunkts tiek izveidots tikai nākamās sistēmas startēšanās laikā, vai arī kad tiek padota speciāla komanda (no sistēmas administratora vai speciāla kontrolpunkta mezgla), lai izveidotu kontrolpunktu.

HDFS veic sistēmas startēšanu pamatojoties uz attēla failā esošo informāciju, *NameNode* inicializē attēla failu no kontrolpuktā esošās informācijas un pēc tam izpilda visas izmaiņas, kas ir ierakstītas žurnāla failā. Kad startēšanās ir pabeigta jauns kontrolfails un tukšs žurnāla fails tiek saglabāti sistēmā un galvenais mezgls var sākt apkalpot lietotāju pieprasījumus. Lai uzlabotu redundanci un drošību kontrolfailu un žurnāla failu kopijas tiek glabātas arī citos serveros – multiplicētas. Tas samazina risku, ka sistēmu nevarēs atjaunot, ja kāds no šiem failiem būs bojāts vai nepieejams.

Visi faili ir sadalīti lietotāja noteiktajos bloka izmēros, pēc noklusējuma bloka izmērs ir 128 MB and tiek glabāti pakļautajos mezglos – *DataNode***.** Tiek veidotas vismaz divas bloku replikas, kas nodrošina redundanci un augstu pieejamību. Ir iespējams mainīt konfigurāciju, lai tiek veidotas arī vairāk par divām replikām, ja ir tāda nepieciešamība. Tā kā vieta, kur glabājas bloka replikas, var tikt mainīta laika gaitā, tad šī informācija nav atainota kontrolpunkta failā.

Lietotāji piekļūst *HDFS* sistēmai lietojot *HDFS* klientu. Tas sākumā pieslēdzas NameNode mezglam un iegūst informāciju par nepieciešamā faila datu bloku atrašanās vietā. Lietotājs lasa bloka saturu no *DataNode*, kas atrodas vistuvāk viņam. Kad dati tiek rakstīti, tad lietotājam sākumā ir jāpieprasa informācija no *NameNode*, kurā no mezgliem dati var tikt ierakstīti. Kad pirmais bloks tiek aizpildīts *NameNode* automātiski padod nākamo, kur var rakstīt. Šie bloki var atrasties dažādos mezglos.

HDFS sistēmai ir izveidotas lietojumprogrammas, kas atrod bloku atrašanās vietas, tas ļauj šādām programmām, kā *MapReduce* veikt uzdevumu datu atrašanās vietā, tādējādi uzlabojot *I/O* veiktspēju. *API* iekļauj sevī arī iespēju uzstādīt replikācijas faktoru katram failam, lai uzturētu failu un bloku integritāti, pēc bloka pievienošanas *DataNode,* tiek izveidoti divi faili, kas attēlo katru no replikām *OS* failu sistēmā. Pirmais fails satur bloka datus un otrs fails satur bloka metadatus, ieskaitot kontrolsummu, katram blokam, kā arī tā veidošanas laiku.

Sākotnēji *HDFS* sistā katrā klasterī bija tikai viens galvenais mezgls, ja tas kaut kādā brīdī kļuva nepieejams, tad visa sistēma arī nebija pieejama. Lai šo situāciju atrisinātu, tika ieviesta *NameNode* replikācija, kā tas ir pakļautajos mezglos. Kas nodrošina augstu pieejamību sistēmai. [1, 2]

Vissvarīgākā sistēmas komponente *HDFS* arhitektūrā ir komunikācija un pārvaldība starp galveno mezglu un pakļautajiem mezgliem. Lai to nodrošinātu tika izveidots speciāls protokols ar sistēmas identifikatoriem un apstiprinājumiem. Sistēmas radīšanas brīdī galvenajam mezglam tiek piešķirts unikāls identifikators. Šis identifikators tiek glabāts

visos klastera mezglos. Līdzīgi katram pakļautajam mezglam tiek piešķirts identifikators tā radīšanas brīdī. Šis identifikators tiek reģistrēts galvenajā mezglā un nekad nemainās, pat ja *DataNode* nomainās *IP* adrese vai piekļuves ports. Sistēmas startēšanās laikā galvenais mezgls pārbauda vai katrs no pakļautajiem mezgliem ir pareizais mezgls, veicot mezglu verifikācijas soļus:

- Katrs pakļautais mezgls nosūta galvenajam mezglam apstiprinājumu ar savu identifikatoru un programmatūras versiju.
- Ja šī informācija nesakrīt ar informāciju, kas atrodas galvenajā mezglā, pakļautais mezgls tiek izslēgts
- Šāda veida verifikācija neļauj nepareizajiem mezgliem pievienoties klasteriem, tādējādi tiek sargāta failu sistēmas integritāte;
- Verifikācijas procesa laikā tiek pārbaudīta arī programmatūras versiju saderība starp galveno mezglu un pakļauto mezglu, jo programmatūras versiju nesaderība var izraisīt datu bojājumus vai zudumus;
- Pēc galvenā mezgla apstiprinājuma un validācijas, pakļautais mezgls nosūta bloka reportu, kas satur bloka identifikatoru, katras bloka replikas garumu un tā radīšanas laiku.
- Pirmais bloka reports tiek nosūtīts uzreiz pēc *DataNode* reģistrācijas galvenajā mezglā;
- Šādi bloka reporti tiek sūtīti galvenajam mezglam katru stundu, kas nodrošina galveno mezglu ar informāciju par bloku repliku atrašanās vietām klasterī;
- Kad jauns pakļautais mezgls tiek pievienots un inicializēts, tam tiek atļauts pievienoties klasterim bez identifikatora (jo tas vēl netika piešķirts), lai saņemtu identifikatoru un reģistrētos galvenajā mezglā.

Savienojums starp galveno mezglu un pakļautajiem mezgliem tiek pārbaudīts katras trīs sekundes – *DataNode* nosūta uz *NameNode* apstiprinājumu par bloku un repliku pieejamību konkrētajā mezglā, kā arī informāciju par pieejamo un izmantoto diska apjomu konkrētajā mezglā, cik daudz datu tiek pārraidīts konkrētajā brīdī. Galvenais mezgls izmanto šo statistiku, lai sabalansētu slodzi starp mezgliem un pārvaldītu failu sistēmā pieejamo vietu. Ja 10 minūšu laikā galvenais mezgls nesaņem no kāda pakļautā mezgla nekādu informāciju par tā stāvokli, tas tiek uzskatīts par nepieejamu, respektīvi arī bloku replikas, kas atrodas šajā mezglā arī ir nepieejamas, tāpēc galvenais mezgls sāk veidot šo bloku replikas citos mezglos, lai nodrošinātu augstu pieejamību.

NameNode var nosūtīt dažādas komandas citiem mezgliem, kā piemēram:

- Sataisīt bloka repliku citos mezglos;
- Izdzēst bloka replikas;
- Pārreģistrēt mezglu;
- Izslēgt mezglu;
- Atsūtīt atskaiti par bloka stāvokli.

Galvenais mezgls ir spējīgs pārvaldīt tūkstošiem šādu komandu sekundē, neietekmējot citas operācijas.

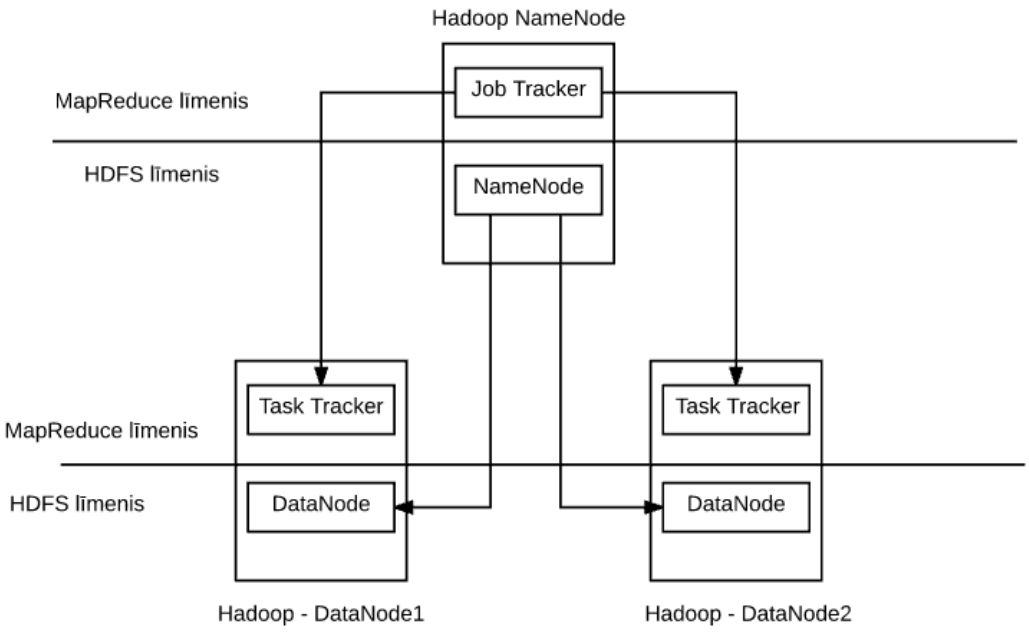
Bez lietotāju pieprasījumu apstrādes un *DataNode* pārvaldības *NameNode* ir arī citas lomas, tās tiek norādītas startēšanas brīdī un var būt – Kontrolpunkta mezgls vai Rezerves kopiju mezgls. Galvenais iemesls kāpēc šādas papildus lomas tika radītas ir atjaunošanās mehānisma radīšana priekš *NameNode*. [1, 2]

Kontrolpunkta mezgls kalpo kā rakstīšanas žurnālā atjaunošanās arhitektūra galvenajam mezglam. Tas savieno esošo kontrolpunktu un žurnālu, lai radītu jaunu kontrolpunktu un žurnālu noteiktajos intervālos un nosūta jauno kontrolpunktu galvenajam mezglam. Kontrolpunkta mezgls strādā uz cita servera nekā *NameNode*, lai nepārslogotu serveri ar galveno mezglu. Pēc kontrolpunkta izveidošanas žurnāls tiek izdzēsts un radīts jauns tukšs žurnāls. Tas ir nepieciešams tāpēc, ka *HDFS* klasteris parasti tiek reti restartēts, līdz ar to žurnāla fails aizņem ļoti daudz vietas un pastāv iespēja, ka tas informācija tajā tiks zaudēta vai bojāta. Kontrolpunkta veidošana noteiktajos intervālos palīdz aizsargāt sistēmu no šāda veida informācijas zuduma un arī paātrina sistēmas atjaunošanos.

Rezerves kopiju mezgls var tikt uzskatīts kā galvenais mezgls, kurš ir tikai priekš lasīšanas. Tas satur visas failu sistēmas metadatus izņemot bloku atrašanās vietas. Tas saņem informāciju par visām sistēmas transakcijām no galvenā mezgla un attiecīgi izmaina savu attēla faila kopiju. Ja kaut kas notiek ar galveno mezglu, rezerves kopiju mezgla attēla fails un kontrolpunkta informācija var tikt izmantota, lai atjaunotu sistēmu līdz stāvoklim kādā tā bija pirms *NameNode* kļuva nepieejams. Rezerves kopiju mezgls var pildīt visas galvenā mezgla funkcijas, izņemot bloku atrašanās vietas pārvaldīšanu un nosaukumu modificēšanu.

Hadoop tika veidots, lai apstrādātu lielus datu apjomus, kā arī lielas caurlaides spējas nodrošināšanai. Pamatā tas ir izstrādāts *Java* valodā tāpēc lielākā daļa procesu, kas tiek izpildīti *Hadoop* sistēmā balstās uz *Java* pamata arhitehtūru un tiek izpildīti *Java* Virtuālās

mašīnas (*JVM*) ietvaros. Attēlā 1.2 ir parādīts kā uzdevumi *Hadoop* arhitektūrā tiek pārvaldīti izmantojot *JobTrackers* un *TaskTrackers* servisus.



1.1. **Uzdevumu pārvaldība *Hadoop* sistēmā**

JobTracker ir serviss *Hadoop* arhitektūrā, kas kontrolē uzdevumu izpildi un pārvaldi. Katrā *Hadoop* klasterī ir tikai viens *JobTracker* process, kas izpilda visus lietotāju uzdevumus un pieprasījumus. *JobTracker* lieto savu *JVM* procesu. Darbu izpildes secība *Hadoop* sistēmā ir šāda:

- Lietotājs iesniedz uzdevuma pieprasījumu;
- Šo pieprasījumu apstrādā *JobTracker* serviss:
  - Tas sazinās ar *NameNode*, lai noskaidrotu nepieciešamo datu atrašanās vietu klasterī;
  - Kad datu atrašanās vieta ir identificēta, tas sazinās ar pieejamo *TaskTracker* mezglu un uztic tam veikt uzdevumu;
  - TaskTracker ziņo *JobTracker* servisam par uzdevuma izpildes status un pabeigšanu.
- Kad uzdevums ir pabeigts, *JobTracker* serviss atjauno uzdevuma status un ziņo lietotājam par uzdevuma izpildi. Lietotājs arī var pieprasīt statusu par darba izpildi tā izpildes laikā un saņemt atbildi no *JobTracker* servisa.

TaskTracker ir mezgls klasterī, kas pieņem uzdevumus no *JobTracker* servisa. Katram tādam mezglam ir noteikts uzdevumu skaits, ko tas spēj vienlaicīgi apstrādāt. Tā pamata īpašības ir:

- *TaskTracker* veido un pārvalda atsevišķus *JVM* procesus, lai izpildītu uzdevumus, ko ir pieprasījis *JobTracker* serviss. *JVM* veidošana palīdz izolēt uzticētos uzdevumus un katra uzdevumu izpildes gaita un rezultāts neietekmē visu *TaskTracker* mezglu;
- *TaskTracker* seko visiem izveidotajiem procesiem, to uzdevumu izpildes gaitai un pieraksta visu procesu izvades un kļūdas paziņojumus. Kad process pabeidz uzdevuma izpildi, *JobTracker* saņem par to paziņojumu;
- *TaskTracker* mezgls sūta regulārus signālus *JobTracker* servisam, lai apliecinātu, ka tas ir pieejams. Šie paziņojumi satur informāciju arī par noslodzi - uzdevumu skaitu, ko var *TaskTracker* uz doto brīdi var uzņemties. Tādējādi *JobTracker* serviss vienmēr zin, kuri mezgli ir pieejami darbu veikšanai.
- Ja *TaskTracker* izpildes laikā saņem kļūdas paziņojumu un informē *JobTracker* servisu par nespēju izpildīt konkrēto uzdevumu ir trīs varianti, kā tas var rīkoties:
    1. Tas var uzticēt uzdevumu citam *TaskTracker* mezglam.
    2. Tas var atzīmēt attiecīgo ierakstu kā nederīgu un izvairīties no šo datu apstrādes.
    3. Tas var atzīmēt konkrēto *TaskTracker* mezglu kā neuzticamu un iekļaut to "melnajā sarakstā".

Šo divu procesu kombinācija ir veids kā *Hadoop* sistēmā tiek izpildīti *MapReduce* procesi. Tā kā *JobTracker* process ir tikai viens, tad tā nepieejamība nozīmē, ka nekādi uzdevumi netiks apstrādāti, tie nonāk rindā uz izpildi, kamēr *JobTracker* serviss atkal kļūst pieejams. [1, 2, 3, 29]

1.2. MapReduce darbības pamatprincipi

MapReduce ir programmēšanas modelis, kas ir paredzēts lielu datu kopu apstrādei, tas sākotnēji tika *Google* kompānijas izstrādāts, lai atrisinātu mērogojamības problēmu mēklēšanas procesos. Tā pamatprincipi ir balstīti uz paralēlo un sadalīto apstrādi bez piesaistes datubāzei. *MapReduce* elastības pamatā ir spēja apstrādāt sadalīto skaitļošanu

lieliem datu apjomiem, kas atrodas klasteros, izmantojot vienkāšus uzdevumu balstītus pārvaldes modeļos.

MapReduce īpašības ir:

- Automātiska paralelizācija;
- Automātiska sadalīšana
- Kļūdu tolerance
- Rīki status noskaidrošanai un uzraudzībai;
- Viekāršs abstrakcijas līmenis programmētājiem;
- Valodas elastība

Valodas pamatā ir fukcionālās programmēšanas modeļi, lielākoties paņemti no *Lisp*. Lietotāju visbiežāk lietotās divas funkcijas ir:

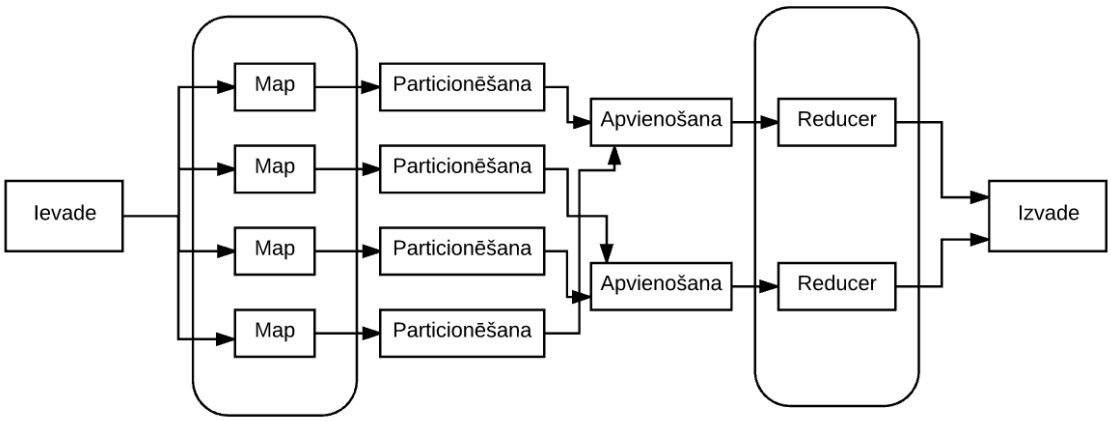
- Map (in_key, in_value) - > (out_key, intermediate_value) list
  - *Map* funkcija, ko uzraksta lietotājs saņem ievades atslēgas un vērtības pāri, un pēc to apstrādes, tiek izveidoti starp atslēgas-vērtības pāri.
  - Bibliotēku funkcijas tiek lietotas, lai sagrupētu visas starp vērtības, kas ir atbilst starpniek atslēgai un pēc tam padod šo rezultātu *Reduce* funkcijai.
- Reduce (out_key, intermediate_value list) - > out_value list
  - Reduce funkcija, ko uzraksta lietotājs saņem starpniek atslēgu un vērtību kopu, kas atbilst šai atslēgai;
  - Tā savieno šīs vērtības veidojot pēc iespējas mazāku vērtīu kopu
  - Izvadē ir nulle vai viena vērtība katram funkcijas izsaukumam
  - Starp vērtības tiek padotas *Reduce* funkcijai, izmantojot iteratoru. Tā ir funkcija, kas ļauj apstrādāt milzīgus vērtību sarakstus, kas nevar tikt izvietoti atmiņā izmēra dēļ vai nevar tikt padoti visi vienlaicīgi.

MapReduce sastāv no dažādu saskarņu bibliotēkām, kā piemēram *Mapper*, *Reducer*, *JobConf*, *JobClient*, *Partitioner*, *OutputCollector*, *Reporter*, *InputFormat*, *OutputFormat*, *OutputCommitter*. [2]

Attēlā 1.3. ir parādīta *MapReduce* arhitektūra. Tās galvenās komponentes ir:

- *Mapper* – savieno ievades atslēga-vērtība pārus starpniek atslēgas-vērtības pāru kopā. Kā ievaddatus tas spēj savienot kā nulli, tā arī daudzus izvades pārus. Pēc noklusējuma mapper veido vienu uzdevumu, kas ir jāizpilda, katrai ievadei;

- *Reducer* – veic vairākus uzdevumus:
  - Šķiro un grupē mapper izvadi;
  - Maina kārtību partīcijām;
  - Veic atkārtotu šķirošanu, ja ir nepieciešams
  - Pāvalda lietotāju noteiktos grupēšanas un particionēšanas noteikumus.



1.3.att. ***MapReduce*** **darbības struktūra**

- *Reporter* – sūta atskaites par uzdevumu pildīšanas progresu, nosūta lietojumprogrammu līmeņa statusa paziņojumus, atjauno lietotāju uzstādītos uzdevuma skaitītājus, apstiprina, ka uzdevumi, kas izpildās ļoti ilgi patiešām tiek pildīti;
- *Combiner* – var tikt lietots, lai veiktu lokālo starp rezultātu agregāciju, kas palīdz pārvaldīt datu apjomu, kas tiek padots no *Mapper* uz *Reducer* komponentēm.
- *Partitioner* – kontrolē atslēgu particionēšanu starprezultātiem. Atslēga (vai atslēgu kopa) tiek lietota, lai atrastu partīciju, pēc noklusējuma partīcijas tiek veidotas izmantojot hash funkciju. Kopējs partīciju skaits ir vienāds ar reduce uzdevumu skaitu konkrētajam procesam.
- Izvades savācējs – savāc izvades no *Mappers* un *Reducers* procesiem;
- Uzdevumu konfigurācija – pamata lietotāju saskarne, ar kuru tiek pārvaldīti *MapReduce* uzdevumi:
  - Parasti to lieto, lai norādītu *Mapper*, *Combiner*, *Partitioner*, *Reducer*, ievade formātu, izvades formātu, un izvades apstiprinājumu katram uzdevumam
  - Ievades failu kopas norādīšana un izvades failu atrašanās vietas norādīšana

    - Var tikt lietota, lai norādītu specifiskus parametrus uzdevumam, kā piemēram, kompresijas izmantošana izvadei vai starprezultātiem;
    - Norāda, cik maksimāli kļūdas ir pieņemamas uzdevuma izpildes laikā, un kāds ir apakšuzdevumu neizpildes pieņemamais procentuālais daudzums.
- Izvades apstiprinātājs – tiek lietots, lai apstiprinātu uzdevumus *MapReduce* programmā. Pamatfunkcijas ir:
    - Norāda uzdevumu, kas ir jāveic inicalizācijas laikā, piemēram, izveidot starpniekdirektoriju, kuru izmantos uzdevuma procesi.
    - Izdzēš nevajadzīgos laicīgos datus pēc uzdevuma izpildes;
    - Pārbauda vai pēc uzdevuma pabeigšanas ir nepieciešams apstiprinājums, ja ir vajadzīgs, tad apstiprina izmaiņas;
    - Ja izpildes laikā radās kļūdas, neapstiprina uzdevuma izpildes rezultātus, atbrīvo servera resursus un izdzēš radītos starprezultātus.
- Uzdevuma ievade:
    - Norāda ievades formātu *Map*/*Reduce* uzdevumam;
    - Validē ievades specifikāciju konkrētajam uzdevumam;
    - Sadala ievades failu vai failus logiskajās daļās, kuras pēc tam tiek iedotas *Mapper* procesam;
- Atmiņas pārvalde, *JVM* lietošana un kompresija tiek pārvaldīta izmantojot uzdevuma konfigurācijas klašu kopu.

MapReduce programmēšana ir balstīta uz funkcionālās programmēšanas pamatprincipiem, kad datuplūsma no viena moduļa notiek līdzīgi tiešā necikliskā grafa principiem, kur vērtību izmaiņas pašreizējā solī izraisa vērtību pārskaitīšanu. Tas nodrošina datuplūsmas konsistenci starp *Mapper*, *Reducer* un *Combiner* moduļiem.

Tā kā *MapReduce* sākotnēji tika veidots, lai atrisinātu meklēšanas problēmu lielās datu kopās, kas nozīmēja miljoniem dokumentu, attēlu, video metadatu caurskatīšanu, kā arī milzīgu lietotāju skaitu, kas veic pieprasījumus dažādās valodās vienlaicīgi, tad *MapReduce* programmas tika veidotas, lai strādātu galvenais-pakļautais režīmā, kur galvenajam ir vienāda izmēra masīvu saraksts, kurus apstrādā vairāki pakļautie procesi paralēli, tiem tiek padotas vienādas masīva daļas. Galvenais process beigās apvieno rezultātu un nosūta to programmas izsaucējam. Šāda veida arhitektūra atbalsta mērogojamību. [2, 10]

MapReduce ir iebūtas daudzas funkcijas, kas ļauj pārvaldīt uzdevumu izpildi un kontrolēt programmas. Ir iespēja veidot arī savas funkcijas un pievienot papildus līmeņus esošai arhitektūrai.

MapReduce galvenais mezgls regulāros intervālos pārbauda vai visi pakļautie mezgli ir pieejami nosūtot signālu, ja tas nesaņem nekādu atbildi noteiktajā laikā, tad uzdevums tiek uzskatīts par neizpildītu un tiek atkārtoti izpildīts uz cita mezgla.

MapReduce ir arī vairāki ierobežojumi:

- Mērogojamība:
  - Maksimālais klastera izmērs – 4000 mezgli;
  - Maksimālais vienlaicīgo procesu skaits – 40000;
- Restartēšana nav vienkārša, jo daudzo procesu stāvoklis ir sarežģīts;
- Ne vienmēr ir vienkārši sadalīt resursus *Map* un *Reduce* daļās. [2, 3, 10]

2. ORACLE PIEDĀVĀTIE RISINĀJUMI BIG DATA APSTRĀDEI

Oracle piedāvā vairākus risinājumus *Big Data* pārvaldei un analīzei, kā piemēram, *Oracle Cloud* servisa izmantošana, kas piedāvā gatavu risinājumu datu glabāšanā un analīzē. Risinājumā ir iekļauti *Big Data* produkti, kas ir nepieciešami datu apstrādei. Šajā gadījumā *Oracle* uzņemas atbildību par sistēmas uzturēšanas tehnisko pusi – datu drošību, sistēmas atjaunojumiem, jaunāko rīku piedāvājumiem, sistēmas korektu darbību un atbalstu problēmsituācijās. Datu glabāšanai mākonī ir arī mīnusi, pirmkārt, neuzticība – daudzas organizācijas negrib izvietot sensitīvos datus ārējos serveros, jo baidās no nevēlamās datu noplūdes. Otrkārt, pakalpojuma cena, kas ir diezgan augsta salīdzinot ar citiem risinājumiem. Treškārt, mazs veiksmes stāstu skaits – lai veiktu datu migrāciju uz mākoni ir nepieciešamas lielas investīcijas, taču ne vienmēr ir redzams vai šie ieguldījumi atmaksāsies.

Oracle ir arī citus produkti, kuri var tikt ieviesti jau esošajās sistēmās, lai palīdzētu apstrādāt un analizēt liela apjoma datus. [33]

2.1. Oracle Big Data Connectors

Oracle Big Data Connectors ir programmatūra, kas integrē *Apache Hadoop* ar *Oracle* datubāzi. Tādējādi organizācijas var lietot *Hadoop* datu apkopošanai un apstrādei un tad savienot šos datus ar uzņēmuma datiem no *Oracle* datubāzes, lai analizētu datus kopumā.

Šis produkts sastāv no piecām pamatkomponentēm:

- *Oracle Loader for Hadoop* (*OLH*) – kas ļauj lietotājiem lietot *Hadoop MapReduce* apstrādi, lai izveidot optimālas datu kopas, kuras pēc tam tiek ielādētas *Oracle* Datubāzē un analizētas. Atšķirībā no citiem produktiem, kas piedāvā datu ielādi no *Hadoop*, tas ģenerē *Oracle* saprotamus iekšējus datu formātus, kas ļauj ielādēt datus ātrāk un izmantot šim procesam mazāk datubāzes sistēmas resursu. *OLH* ir pievienots kā pēdējais solis *MapReduce* transformācijā, kā atsevišķs map – particionēšanas – reduce solis. Tajā tiek izmantoti *CPU* resursi no *Hadoop* klastera, lai pārveidotu datus *Oracle* vajadzīgajā formātā, kas ļauj samazināt *CPU* patēriņu datubāzes serverī. Kad dati ir ielādēti datubāzē, tie ir pieejami lietotājiem gan pa tiešo veicot *SQL* pieprasījumus, gan lietojot dažādus *Business Intelligence* (*BI*) rīkus.

- *Oracle SQL Connector* priekš *HDFS* – nodrošina augsta ātruma savienojumu starp *HDFS* atrodamiem datiem un *Oracle* datubāzi. Šis savienotājs ļauj lietotājiem veikt pieprasījumus datiem pa tiešo no *HDFS*, atbilstoši viņu vajadzībām. Tas tiek nodrošināts veidojot ārējo tabulu *Oracle* datubāzē, kas ļauj tiešo *SQL* piekļuvi *HDFS* datiem. Visi datu pieprasījumi var tikt veikti izmantojot *SQL* valodu, apvienoti ar datiem, kas atrodas *Oracle* datubāzē *SELECT* pieprasījumā vai arī dati no *HDFS* var tikt ielādēti datubāzē. Piekļuve *HDFS* datiem ir optimizēta ātrai datu pārvietošanai un paralelizācijai, izmantojot automātisko slodzes balansēšanu. Dati *HDFS* sistēmā var būt Oracle *Data Pump* rīka veidoti vai citā *Oracle* saprotamā formātā.
- *Oracle Data Integrator Application Adapter* priekš *Hadoop* – palīdz vienkāršot integrāciju starp *Hadoop* un *Oracle* datubāzi. Kad dati ir pieejami datubāzē, gala lietotāji var izmantot *SQL* un *Oracle BI* rīku, lai piekļūtu datiem. Uzņēmumi, kuri jau lieto *Hadoop* risinājumu un nevēlas izmantot *Oracle* risinājumus kā *Oracle Big Data Appliance* datu glabāšanai var izmatot šo moduli, lai izveidotu vienotu programmatūras risinājumu.
- *Oracle R Connector* priekš *Hadoop* – ir programmatūra, kas ļauj piekļūt *Hadoop* sistēmai un datiem, kas ir izvietoti *HDFS*. *R Connector* piedāvā lietotājiem atvērtā koda statistikas vidi *R* ar iespēju analizēt datus, kas atrodas *HDFS*, kā arī mērogojamību – iespēju izpildīt *R* modeļus lielajiem datu apjomiem, izmantojot *MapReduce* apstrādes metodes. *R* lietotājiem nav jāapgūst papildus lietojumprogrammatūra vai valoda, lai to darītu, tie var lietot vairāk nekā 3500 atvērtā koda *R* pakotnes datu analīzei *HDFS* sistēmā. Šis produkts var tikt lietots arī ar *Oracle Advanced Analytics* iespēju *Oracle* datubāzē. Tā ļauj *R* lietotājiem strādāt ar datubāzes datiem neapgūstot *SQL* valodu vai datubāzes darbības konceptus. Visa datu apstrāde notiks pa tiešo datubāzē, kas ļoti vienkāršo *R* sistēmas lietotāju darbu.
- *Oracle XQuery* priekš *Hadoop* – ļauj lietot *XQuery*, lai apstrādātu un transformētu tekstu, *XML, JSON* un *Avro* formātos glabāto informāciju *Hadoop* sistēmām. Tas ļauj izmantot Hadoop piedāvāto paralēlās apstrādes iespēju, lai apstrādātu *W3C XQuery* izteiksmes, lietojot visus klaster mezglus. Šis rīks palīdz pārvietot *XQuery* apstrādi pa tiešo datu atrašanās vietā, nevis pārvietot

datus uz *XQuery* programmatūras atrašanās vietu. Pēc tam rezultāti var tikt ielādēti datubāzē, ja ir tāda nepieciešamība. [22, 29, 33]

2.1. Oracle Big Data Discovery

Oracle produkts, kas ļauj piekļūt pa tiešo *Hadoop* sistēmā esošajiem datiem, kā arī vizualizēt un analizēt datus. Tas ļauj atrast vajadzīgos datus ātri, ielādēt datus no *Excel* vai *CSV* formātiem pa tiešo uz *Hadoop* sistēmu. Šī programma tika izveidota, lai atrisinātu vairākas lietotāju problēmas:

- Lietotāju saskarnes vienkāršošana – iespēja vairāk lietotājiem izmantot *Big Data* iespējas, bez sarežģītas priekšapmācības;
- Datu sagatavošana pirms lietošanas, tiek samazināts laiks, kas nepieciešams datu kopas sagatavošanai, šo laiku lietotāji var veltīt datu analīzei un citu biznesa problēmu risināšanai;
- Iespēja ielādēt jaunus datus jau esošajos analītiskajos projektos, pievienot jaunākos datus jau ielādētajiem.

Viena no *Big Data Discovery* komponentēm ir Studio – tā ir lietotāju saskarne, kas ļauj lietotājiem bez priekšzināšnām veikt dažādas darbības ar datiem. Tā ir veidota, lai palīdzētu lietotājiem atrast visu vajadzīgo informāciju – ir pārdomāta navigācijas sistēma, kā arī daudzas datu vizualizācijas iespējas, kā piemēram, grafiku veidošana, kartes, pivot tabulas, datu apkopošana, laika joslas. Ir iespējams veidot arī savas vizualizācijas komponentes.

Studio ietilpst rīks datu kopu ielādēšanai, pārskatīšanai, atjaunošanai un transformēšanai. Tas ļauj veidot projektus ar vienu vai vairākām savienotām datu kopām, ielādēt papildus datus esošajos projektos, rakstīt transformācijas skriptus, ja ir nepieciešams. Ir iespēja arī veidot un piešķirt lietotājiem lomas, lai kontrolētu to piekļuvi dažādiem datiem.

Datu apstrādes komponente izpilda procesu kopas un uzdevumus, to mēdz saukt arī par datu apstrādes darba plūsmu. Liela daļa šo apstrādes procesu notiek *Hadoop* sistēmā. Šīs komponentes ziņā ir kontrole par dažādām darba plūsmām sistēmā, kā piemēra, datu ielādi, datu atjaunošanu, datu transformēšanu. Tā automātiski atrod jaunas *Hive* tablas un

ielādē *Big Data Discovery* sistēmā. Kā arī veic inkrementālu datu atjaunošanu tabulās un uztur *Big Data Discovery* datu kopas sinhronizētas ar *Hive* tabulām, kuras tika radītas.

Piemēram, datu ielādes darba plūsmā datu apstrādes komponente veic šādus uzdevumus:

- Atrod datus *Hive* tabulās
- Rada *Big Data Discovery* datu kopas
- Izpilda dažādas operācijas atrastajās datu kopās;
- Indeksē datu kopas.

Datu apstrādes komponentei ir arī komandrindas saskarne, kas ir *Linux shell* programma, kas startē datu apstrādes darba plūsmas *Hadoop* sistēmā. Ir iespējams kontrolēt tās soļus un darbības. Komandrindas saskarnei var piekļūt gan manuāli, gan izmantojot cron uzdevumu. Saskarne ļauj izpildīt visas tās pašas darbības kā grafiskā lietotāju saskarne, kā arī dod lielāku kontroli par procesiem un iespēju veidot elastīgākas komandas datu apstrādei. Taču, lai lietotu šo saskarni ir nepieciemas priekšzināšanas.

Vēl viena sistēmas komponente ir *Dgraph*, kas izmanto datu struktūras un algoritmus, lai radītu reālā laika atbildes uz lietotāju pieprasījumiem. Tas glabā datu indeksus, kad tiek saņemts pieprasījums no lietotāja, *Dgraph* tos lieto, lai ātri atrastu rezultātu. *Dgraph* ir bezstāvokļa arhitektūra, tas nozīmē, ka katram pieprasījuma gadījumā tiek nosūtīts pilns pierasījuma teksts uz *Dgraph* komponenti. Tas ļauj konfigurēt vairākas *Dgraph* instances, lai nodrošinātu slodzes sadalījumu starp tā un redundanci. Katra no instancēm darbojas neatkarīgi no citām.

Big Data Discovery klasteris var saturēt divas vai vairāk *Dgraph* instances, tās nodrošina atbildes uz gala lietotāju pieprasījumiem un pārvalda indeksus kopīgajā disku masīvā. Viena no *Dgraph* instancēm ir galvenā – tā pārvalda rakstīšanas un datu atjaunošanas operācijas, savukārt pārējās atbild uz datu lasīšanas pieprasījumiem.

Dgraph instances parasti darbojas uz atsevišķa servera, nevis *Hadoop* servera. Piekļuvi tām kontrolē *Dgraph Gateway*, kas ir uz *Java* valodas balstīta programmatūra, kuru pārvalda *Weblogic* serveris. Tā ļauj savienot *Studio* komponenti un *Dgraph* instances. [23, 24]

2.3. Oracle GoldenGate priekš Big Data

Oracle GoldenGate priekš lielajiem datiem ir produkts, kas ļauj sūtīt transakcijas datus lielo datu sistēmās reālajā laikā, neietekmējot datu avota sistēmas veiktspēju. Produkts piedāvā sūtīt datus uz dotajā brīdī populārākājiem lielo datu risinājumiem, tādiem kā *Apache Hadoop, Apache Hbase, Apache Hiv*e un *Apache Flume*.

Tas palīdz integrēt dažādus produktus reālajā laikā, tādējādi analītiskās sistēmas saņem datus bez nobīdes laikā, kā arī ļauj veidot aktīvu datubāzes replikāciju, kas dod iespēju datubāzes migrācijai bez tās izslēgšanas, kas ir īpaši svarīgi augstas pieejamības sistēmās.

Sistēmā ir iekļauts arī *Oracle GoldenGate* priekš *Java*, kas ļauj viegli integrēt sistēmu arī ar citām lielo datu sistēmām, tādām kā *Oracle NoSQL, Apache Kafka, Apache Storm, Apache Spark* un citām.

Oracle GoldenGate piefiksē datus no heterogēnā avota (ieskaitot *Java* bāzētas ziņojumu sistēmas) neagresīvi, radot pēc iespējas mazu papildus slodzi sistēmā. Tas glabā datubāzes transakcijas tā saucamos *Trail* failos un ielādē tos *Java* adapterī. Tajā atrodas atbilstoši adapteri dažādām lielo datu sistēmām, kā piemēram, *HDFS* adapteris un citi, pēc tam šie dati tiek sūtīti uz atbilstošo sistēmu. Sūtīšana notiek reālajā laikā. Sūtīšana var notikt arī uz vairākām sistēmām vienlaicīgi, ir iespējams arī nokonfigurēt sūtīšanas veidu, vai visi dati tiek sūtīti nepārtraukti vai arī sadalīti noteiktajās datu porcijās.

Transakciju dati var tikt sūtīti ne tikai uz citām relāciju datubāzēm, bet arī uz citām lietojumprogrammatūrām, *ETL* rīkiem, *JMS* ziņojumu sistēmām, *Big Data* sistēmām. [26, 33]

2.4. Oracle Big Data SQL

Oracle produkts, kas palīdz veikt pieprasījumus dažādiem datu avotiem, izmantojot tikai *SQL* pieprasījumus. Ir optimizēts, lai lietotu sadalītajās sistēmās. *Oracle Big Data SQL* atbalsta dažādus datu avotus, kā piemēram, *Hadoop, NoSQL* datubāzes un *Oracle* datubāzes. Automātiski veido *Oracle* datubāzes ārējās tabulas, kuras palīdz apvienot datus no dažādiem avotiem. Arhitektūrā ietilpst arī *SmartScan* funkcija priekš *Hadoop*, kas palīdz minimizēt datu pārsūtītšanu starp sistēmām, jo atlasa tikai nepieciešamos datus, kas palīdz palielināt sistēmas kopējo ātrdarbību un veiktspēju. Lietotājam nav jāmācās papildus

valodas, lai veiktu pieprasījumus *NoSQL* vai *Hadoop* sistēmās, jo visi pieprasījumi tiek veidoti *SQL* valodā.

Vienota *SQL* pieprasījuma nodrošināšana tiek panākta veidojot sava veida paplašinājumu *Oracle* datubāzes ārējo tabulu funkcionalitātei. Šie paplašinājumi tika izveidotim ņemot vērā citu sistēmu formātus, kā piemēram, *Hadoop* datu ievades formātus un *Hive SerDe* saskarnes. Tādējādi lietotājam dati no dažādām lielo datu sistēmām tiek parādīti datubāzes tabulās, bet visas pieejas un valodu semantika tiek automātiski nodrošināta pieprasījuma laikā. Uz ārējām sistēmām darbojas speciāli aģenti, kas veic datu apstrādi ārējās sistēmās un nosūta uz *Oracle* datubāzi tikai nepieciešamos rezultējošos datus. Tas palīdz samazināt pārsūtīto datu apjomu un palielināt sistēmas veiktspēju. Aģenti transformē datus vajadzīgajos formātos pirms nosūtīšanas, kā arī nodrošina vajadzīgos drošības standartus.

SmartScan funkcija novērtē datus, lai tiktu nosūtīti tikai pieprasījumam nepieciešamie dati, kā piemēram:

- Novērtē *SQL* pieprasījuma *WHERE* daļu
- Kolonnu projekcija
- Pielieto vajadzīgos filtrus labākai *JOIN* darbībai;
- Datu pārsūtīšanai starp sistēmām izmanto Blūma filtrus;
- *JSON* standarta atbalsts un datizraces modeļu novērtēšana.

Pēc *SmartScan* apstrādes dati tiek pārveidoti vajdzīgajos formātos un nosūtīti uz *Oracle* datubāzi. Datubāzē tie tiek apstrādāti tālāk – pielietotas vajadzīgās *join, union, PL/SQL* funkcijas. [33, 37]

3.CITU PIEGĀDĀTĀJU PIEDĀVĀTIE RISINĀJUMI

Bez *Oracle* ir pieejami arī citu piegādātāju risinājumi *Big Data* datu pārvaldei un analīzei. Saskaņā ar *Gartner* 2016.gada pētījumu par Datu noliktavu un Datu pārvaldes un analīzes risinājumiem [11] līderi ir *Oracle, Teradata, Microsoft, IBM* un *SAP*. Pētījumā tika salīdzināti 21 programmatūras piegādātāji pēc 15 dažādiem kritērijiem. Rezultāti tika apkopoti arī grafiski izmantojot kvadrantu, kurā visi piegādātāji tika sadalīti 4 kategorijās – līderi, izaicinātāji, nišas spēlētāji un sapņotāji. Attēlā 3.1. ir parādīts *Gartner* pētījuma rezultātu apkopojums.

3.1.att. ***Gartner*** **kvadrants [11]**

Savā pētījumā *Gartner* definē arī jaunu terminu loģiskās datu noliktavas – *LDW,* kas ir datu noliktavas, kas lieto repozitorijus, virtualizāciju un sadalīto datu apstrādi savā arhitrektūrā,

datu noliktavas vairs nav tikai integrēta relāciju datubāze. Pētījumā tiek norādīts arī, ka arvien vairāk tiek piedāvāti un izmantoti mākoņpakalpojumu analīzes sistēmu izveidošanā. [11]

3.1. Teradata piedāvātie risinājumi

Teradata piedāvā vienotu arhitektūru – *Teradata Unified Data Architecture*, kas ļauj piekļūt un pārvaldīt datus no dažādiem avotiem, gan *Hadoop*, gan tradicionālām *RDBMS* datu noliktavām, apvienojot visu vienā datu pārvaldes sistēmā.

Apvienotās arhitektūras komponentes ir *Teradata QueryGrid*, *Teradata Listener*, *Teradata Unity* un *Teradata Viewpoint*. [30]

Teradata QueryGrid™ ļauj piekļūt dažādiem datu analīzes rīkiem, apvienojot dažādus rīkus vienotā vidē. Lielākā priekšrocība – iespēja ar vienu *SQL* pieprasījumu iegūt datus no dažādām platformām un datu avotiem. *QueryGrid* ļauj piekļūt arī *Hadoop* datiem, tas tiek realizēts izmantojot speciālo konektoru priekš *Hadoop* – *Teradata Connector* as ļauj aplikācijām vienotajā datu arhitektūrā apmainīties ar datiem savā starpā. Tas ir ātrdarbīgs, paralēls konektors, kas savieno *Teradata* un *Hadoop,* ieskaitot *Hortonworks*, *Cloudera* un *MapR*. [14, 18]

Teradata QueryGrid™ ļauj automatizēt un optimizēt analītikas sistēmas, jo tiek:

- Minimizēta datu pārvietošana starp sistēmām, dati tiek apstrādāti to atrašanās vietā;
- Minimizēta datu duplicēšana
- Automatizēts datu analīzes process starp sistēmām;
- Atļauta divvirzienu datu pārvietošana.

Teradata Listener™ ir risinājums, kurš tika izveidots, lai ļautu programmētājiem vienkāršot datu plūsmu starp dažādām sistēmām veidošanu un uzturēšanu. Tas darbojas kā centrālais serviss, kas raksta un uztur simtiem plūsmu vienotajā arhitektūrā. Nodrošina arī datu pārraidi bez zaudējumiem un zemu datu aizturi, kas ļauj nodrošināt tuvu reālā laika datu piegādi dažādām lietojumprogrammām. [14]

Teradata Unity ir četru *Teradata* produktu apvienojums integrētajā sistēmā, kas ļauj analizēt datus. Komponentes, kas ietilpst šajā sistēmā, ir:

- *Unity Director* - sinhronizē datubāzes savā starpā, nodrošinot lietotāju un pieprasījumu koordinēšanu starp sistēmām un vienotu skatu uz datiem dažādās datubāzēs;
- *Unity Loader* – palīdz ielādēt datus sistēmā no dažādiem datu avotiem un nodrošina datu atjaunošanu, atbilstoši vajadzībām. Spēj nodrošināt datu ielādi no dažādiem avotiem un ir elastīgs lietošanā, līdz ar to izslēdz vajadzību veidot papildus risinājumus datu ielādei;
- *Unity Data Mover* – ļauj pārvietot un kopēt datus starp dažādām sistēmām, strādā ar Teradata, Aster Discovery un Hadoop;
- *Unity Ecosystem Manager* – palīdz izveidot vienotu pārvaldes sistēmu *Teradata* integrēto datu noliktavām, *Teradata Aster Discovery* platformām un *Hadoop*. Kā arī monitorēt sistēmu stāvokli, izsekot operācijas un atkarības starp sistēmām, un nodrošināt lietotājus ar informāciju par viņu pieprasījumu stāvokli, ieskaitot latences problēmas.

Teradata Viewpoint savukārt ļauj gan vienkāršiem lietotājiem, gan sistēmas administratoriem izveidot persosnalizētu paneli ar rīkiem, kas ļauj pārvaldīt un pārskatīt *Teradata* sistēmas. Piekļuve šim panelim ir iespējama caur pārlūkprogrammu. *Teradata Viewpoint* ir izveidots kā pašapkalpošanās rīks, kas atvieglo lietotāju piekļuvi sistēmai, tā tiek nodrošināta caur interneta pārlūkprogrammu, saskarne ir grafiska un intuitīvi saprotama lietotājiem. Tajā ir iespējams iekļaut tikai lietotājam vajadzīgos rīkus un sistēmas, tādējādi vienkāršojot un atvieglojot lietotāju darbu. [14, 18]

3.2. IBM piedāvātie risinājumi

IBM piedāvātie *Big Data* risinājumi balstās uz *Hadoop* sistēmu un ļauj lietotājiem glabāt, analizēt un pārvaldīt datus no dažādiem datu avotiem. *IBM* piedāvā gan produktus, kas var tikt ieviesti esošajās arhitekturās, gan pakalpojumus, piemēram, *Hadoop-as-a-Sevice* pakalpojums *IBM* mākonī.

Viens no piedāvātajiem risinājumiem ir *Hadoop* sistēma, kas ir piemērota liela apjoma datu analīzei, kas var būt glabāti gan strukturētā, gan nestrukturētā veidā. Dati var tikt analizēti nemainot to formātu. Šajā risinājumā ietilpst *IBM® BigInsights™* produkti:

- *IBM BigInsights BigIntegrate* – datu integrācijas rīks, kas nodrošina savienojumu, datu transformāciju un datu pārsūtīšanu starp datu mezgliem *Hadoop* klasterī. Rīks

ļauj izveidot mērogojamu, *in-memory* datu integrācijas platformu, kas darbojas *Hadoop* klasterī. Kā arī nodrošina metadatu pārvaldīšanu un datu profilu veidošanu Lai nodrošinātu reālā laika datu analīzes procesu *BigInsights BigIntegrate* ir integrēts ar *IBM* Streams risinājumu.

- *BigInsights BigQuality* – produkts, kas nodrošina *Hadoop* klasterī lielu datu kopu profilēšanu, attīrīšanu un monitorēšanu. Tas ļauj piemērot datu kvalitātes prasības *Hadoop* datiem, atbalsta datu maskēšanu un testa datu pārvaldīšanu automātiski atpazīstot sensitīvo un personīgo informāciju, kas tiek glabāta *Hadoop* sistēmā. Identificē datu tipus kolonnas, ieskaitot kredītkaršu informāciju, identifikācijas numurus, telefona numurus. Palīdz standartizēt un verificēt datus (piemēram, adreses) *Hadoop* klasterī.
- *IBM BigInsights for Apache Hadoop – Hadoop* risinājums, kas palīdz pārvaldīt un analizēt liela apjoma datus. Tajā ietilpst *IBM BigInsights Data Scientist* modulis, kas ļauj analizēt datus, kā arī moduļi, kas palīdz vizualizēt datus un optimizēt dažādu analītikas funkciju izmantošanu. To nodrošina *Big SQL* (ļauj izmantot *SQL* pieprasījumus strādājot ar *Hadoop*) un *BigSheets* (veic datu vizualizāciju un manipulāciju ar *Hadoop* datiem) produkti.
- *IBM InfoSphere Big Match for Hadoop* – palīdz analizēt liela apjoma strukturētos un nestrukturētos datus. Ietver sevī teksta analīzes iespējas un varbūtiskās apvienošanas metodes, lieto statistikas algoritmus, lai savienotu datus pēc iespējas precīzāk, kā arī izmanto pastāvīgu glabātuvi, kurā tiek ievietoti savienotie identifikatori. Nodrošina *API* atbalstu, ieskaitot *Java* un *REST* balstītus *API*. Piedāvā konfigurējamu algoritmu bibliotēku, kas ļauj lietotājiem vieglāk atrast vajadzīgās funkcijas. Rīks izmanto *MapReduce* sadalīto datu apstrādi, lai vajadzīgo datu kopu apvienošana notiktu ātri.

IBM stream risinājums dod iespēju analizēt datus reālajā laikā, kas ir īpaši svarīgi, ja datu pieaugšanas ātrums ir liels un tie ātri noveco. Risinājums izmanto teksta analīzes metodes un ģeotelpisko datu analīzi. Dati, kas var tikt analizēti, var būt dažādos formātos, ieskaitot teksta datus, attēlus, audio, video, balss, tīkla trafika dati, epasti, *GPS* dati, finanšu transakcija vai sensoru traces faili. Platforma iekļauj sevī rīkus, kas spēj reālā laikā analizē telekomunikāciju datus un sociālo mēdiju datus. Kā arī spēj filtrēt datus un atstāt tikai nepieciešamos, kas ļauj samazināt nepieciešamās diska vietas apjomu. *IBM Stream* pielāgojas esošajiem datu tipiem un formātiem, tāpēc platformas pielietošanas iespējas ir

ļoti plašas. Ir atbalsts arī *Java* un *C++* koda lietošanai, kā arī *Predictive Model Markup Language* (*PMML*) modeļiem. To var integrēt gan ar *IBM* produktiem kā *DB2, IBM Infomix, IBM PureData System*, gan ar citu piegādātāju produktiem, kā piemēram, *Oracle, Microsoft SQLServer, MySQL* un citiem. [19]

3.3. Microsoft piedāvātie risinājumi

Microsoft liek uzsvaru uz *Big Data* risinājumiem mākonī. *Microsoft Azure HDInsight* serviss piedāvā iespēju maksāt par *Hadoop* bāzētu datu apstrādi, tā pamatā ir *Azure* virtuālo mašīnu klasteris, uz kuriem darbojas *Hortonworks Data Platform* (*HDP*), kā arī ir izveidota integrācija ar *Azure blob* datu glabātuvi.

Big Data risinājumos dati parasti tiek glabāti failos uz diska, savukārt *HDInsight* risinājumā šie faili tiek glabāti *Azure blob* formā. Tiek nodrošināta pilnīga savienojamība ar *Hadoop* failu sistēmu – komandas un procesi, kas darbojas *HDFS* sistēmā var tikt palaisti arī *Azure blob* sistēmā. Tādējādi *Azure blob* datiem var piekļūt gan caur *HDFS*, gan izmantojot standarta piekļuves iespējas. Ir iespēja arī izveidot klasteri, kurā atrodas *HBase* atvērtā koda datu pārvaldes sistēma. *Hbase* ir *NoSQL* datu glabāšanas sistēma, kas nodrošina datu apstrādi un glabāšanu *Hadoop* klasterī. [21, 32]

HDInsight atbalsta lielu daļu *Hadoop* pieprasījumu, transformāciju un analīzes rīku, kā arī var tikt pievienoti citi rīki, ja ir vajadzība. Piemēram, tiek atbalstīti:

- *Hive* – kas ļauj izmantot *SQL* veida valodu *HiveQL*, lai rakstītu pieprasījumus un veiktu dažādas manipulācijas ar datiem. Piemēram, izmantot *CREATE TABLE* komandu, lai izveidotu tabulu, vai lietot *SELECT* pieprasījumus, lai atlasītu vajadzīgos datus;
- *Pig* – ļauj piekļūt datiem izmantojot augsta līmeņa valodu *Pig Latin*;
- *Map*/*reduce* – var tikt izmantotas *Java* valodā rakstītas komponentes, kas tiek izpildītas pa tiešo *Hadoop* sistēmā, kā arī var tikt lietota *Hadoop streaming* saskarne, kas ļauj izpildīt *map* un *reduce* komandas rakstītas citās valodās.
- *Mahout* – ļauj veikt datizraces pieprasījumus un atrast vajadzīgā tipa informāciju no failiem. Kā piemēram, atrast kam lietotāji dod priekšroku, izvērtējot to uzvedību, vai grupēt dokumentus ar līdzīgu saturu, klasificēt dokumentus;
- *Storm* – sadalīta reālā laika skaitļošanas sistēma, kas ļauj ātri apstrādāt lielas datu plūsmas. Tā ļauj veidot kokus un vērstos acikliskos grafus (*DAG*), kas ahinhroni

apstrādā datus, izmantojot lietotāju definēto paralēlo procesu skaitu. Šāda sistēma var tikt izmantota reālā laika datu analīzei, *ETL* procesiem.

HDInshigt ir pieejami arī dažādu veidu rīki, kas palīdz savienoties ar citām sistēmām un veikt dažādu datu apstrādi:

- *ODBC draiveris*, kas ļauj savienot datubāzess vai vizualizācijas rīkus ar datiem, kas atrodas *Hive* tabulās;
- *Hcatalog* – var tikt izmantots kopā ar *Hive* un *Pig* pieprasījumiem, lai izveidotu abstrakcijas slāni, kas ļauj aizvietot failu fizisko atrašanās vietu ceļus uz vienkāršākiem un intuitīvi saprotamākiem nosaukumiem, tādējādi atvieglojot pieprasījumu rakstīšanu un failu pārvaldi;
- *Sqoop* – ļauj veikt datu importu un eksportu no relāciju datubāzēm uz *HDInsight* un otrādi;
- *Oozie* – nodrošina darba plūsmas un operāciju automatizēšanu.

HDInsights piedāvā lietotājiem arī grafisko saskarni – pārvaldības paneli, kas ļauj gan pārvaldīt klastera resursus, gan izpildīt vajadzīgās manipulācijas ar datiem, redzēt darbu statusus un problēmas. [12, 21]

3.4. SAP piedāvātie risinājumi

SAP piedāvā gan mākoņpakalpojumus, gan produktus datu analīzei, gan *in-memory* datu bāzi.

SAP HANA ir in-*memory* platforma, kas tika izveidota, lai risinātu *Big Data* problēmas. *SAP HANA* datubāze atbalsta gan *OLTP*, gan *OLAP* procesus, tā ir piemērota, lai paralelizētu procesus un glabātu lielu datu apjomu atmiņā vai diskā, kur dati tiek uzglabāti kolonnās. Tiek nodrošināta integrācija ar *R* serveri, tādējādi dodot iespēju analizēt izmantojot *R* scriptus. Dati var tikt glabāti dažādos formātos, kā piemēram *PDF* failos, *HTML, RTF, MSG, Microsoft Office* dokumentos, ir atbalsts dažādām valodām (kopā 32), ar kuru palīdzību var veikt vajadzīgo datu meklēšanu, tajā skaitā arī *SQL* veida valodai. [35, 36]

Viens no pamatproduktiem, ko piedāvā *SAP* priekš liela apjoma datiem ir *SAP HANA Vora*, kas ir *in-memory* pieprasījumu instruments, kas palīdz analizēt datus no dažādiem datu avotiem. Šis produkts paplašina *Apache Spark* rīku, lai nodrošinātu interaktīvu

Hadoop datu analīzi. Tas tika izveidots, lai atrisinātu pamata problēmas, ar ko saskaras *Big Data* apstrādē:

- Ļauj analizēt datus gan no relāciju datubāzēm, gan no ārējiem avotiem ar nestrukturētiem datiem;
- Nav nepieciešami integrācijas risinājumi, jo dati var tikt apstrādāti pa tiešo *Hadoop* klasterī;
- Dod iespēju analizēt liela apjoma *Hadoop* datus reālā laikā. [28, 36]

SAP ir izveidojis sadarbību ar *Cloudera, Databricks, Hortonworks, MapR Technologies* un citiem tehnoloģiju ražotājiem, lai paplašinātu *SAP HANA Vora* rīka iespējas. Pēc būtības *SAP HANA Vora* darbojas kā sava veida konektors starp *Hadoop* un *SAP HANA* datubāzi. [28]

3.5. Piedāvāto risinājumu salīdzinājums

Katram ražotājam ir sava pieeja *Big Data* apstrādes un analīzes problēmu risinājumiem, taču ir arī kopīgas tendences, kā piemēram, visi risinājumu piegādātāji piedāvā izvietot infrastruktūras izvietošanu mākonī, kas ļauj ieekonomēt uz aparatūras iegādes rēķina un tā uzturēšanas un atbalsta darbu izmaksām. Kopīgs ir arī tas, ka visi risinājumi balstās uz *Hadoop* tehnoloģijas integrāciju, apvienošanu ar esošajiem produktiem. Ir bijuši arī mēģinājumi apiet *Hadoop* un izveidot savus risinājumus lielo datu apjomu apstrādei. Kā viens no tādiem ir eksperimentiem ir bijis *Oracle* – tika mēģināts apvienot *Hadoop* un tradicionālo *RDBMS* datubāzi. Parasti pieprasījumi tiek kompilēti kā *MapReduce* uzdevumu secība un tie tiek izpildīti *Hadoop* klasterī. Eksperimentā tika izveidots bibliotēkas prototips, kas ļāva izpildīt *Hadoop MapReduce* komandas *Oracle RDBMS* sistēmas iekšienē, tie tika izpildīti izmantojot *Oracle JVM*. Galvenie šī risinājuma ieguvumi bija pilnīga datubāzes savienojamība ar *Hadoop* – bija iespējams izpildīt *Hadoop* programmas relāciju datubāzē neko nemainot kodā. Šī ir būtiska atšķirība no citiem risinājumiem, parasti tiek piedāvāts veidot *SQL* pieprasījumus un tie tiek pārveidoti uz *MapReduce* komandām. Kā risinājuma priekšrocības tiek uzskatītas arī datu atrašanās vienotā infrastruktūrā – nav nepieciešams pārsūtīt datus vai daļu no datiem citur, lai veiktu apstrādi. Arī neatkarība no *Hadoop* klastera, visas darbības tiek izpildītas *Oracle RDBMS* datubāzē. Risinājums ļauj arī iekļaut *MapReduce* funkcijas *SQL* pieprasījumos, tādējādi ir

iespējams izmantot abas valodas, kā arī valodu miksējumu, lai veiktu datu apstrādi un manipulācijas. [4, 8, 31]

Taču vēlākie pētījumi [6] parādīja, ka salīdzinājumā ar *Hadoop* klasteri *MapReduce* realizācija relāciju datubāzes iekšienē nav efektīva. 3.1. tabulā ir apkopotas galvenās atšķirības starp *Hadoop* klasteri un *In-database MapReduce* realizāciju, kas tika eksperimentu un salīdzinājumu rezultātā. Viena no būtiskākajām atšķirībām – liela datu apjoma gadījumā *RDBMS* sistēmā apstrāde notiek lēnāk, kā arī nav aizsardzības pret aparatūras un programmatūras bojājumiem, ja *HDFS* klasterī kāds no mezgliem pārstāj strādāt, visas klastera darbība netiek pārtraukta. *Oracle RDBMS* gadījumā šis nosacījums ir grūtāk realizējams, nepieciešams ieviest papildus risinājumus, kā piemēram, *Oracle RAC – Real Application Cluster*, kas ļauj datubāzi sadalīt pa vairākiem serveriem un nodrošina tās pieejamību, taču šāda veida risinājums ir dārgs, jo ir nepieciešamas papildus licenses.

3.1.tabula

***In-database MapRedu* un *Hadoop* klastera salīdzinājums [6]**

Oracle In-database MapReduce	***Hadoop* klasteris**
Apstrādā tikai strukturētos datus	Apstrādā gan strukturētos, gan nestrukturētos datus
Dati tiek apstrādāti vienuviet, to atrašanās vietā	Sadalītā datu apstrāde
Nav aizsardzības mehānisma pret aparatūras vai programmatūras bojājumiem	Ir nodrošināti risinājumi nepārtrauktai darbībai aparatūras vai programmatūras bojājumu gadījumos.
MapReduce pieprasījumi ir sarežģīti	*Pig* skripti ir vienkārši un intuitīvi saprotami
Liela apjoma datu apstrāde ir laikietilpīga	Liela apjoma datu apstrādē neaizņem daudz laika
Datu analīze un apstrāde vairāk piemērota maza apjoma datiem	Datu analīze un apstrāde vairāk piemērota liela apjoma datiem

Tikai *MapReduce* funkcionalitāte ir pieejama	Gan *MapReduce*, gan *HDFS* funkcionalitāte ir pieejama

Tā kā relāciju datubāzēm un *Hadoop* klasteriem ir dažāds darbības veids un pielietojums, tad ražotāji koncentrējas uz šo sistēmu apvienošanu, vienotas arhitektūras izveidošanas, kā to dara, piemēram, *Teradata* vai integrācijas risinājumu izveidošanu, kas ļautu paņemt labākās īpašības no katras sistēmas un mazināt vājo vietu ietekmi.

Lielākā daļa ražotāju ir izveidojusi konektorus, kas ļauj savienot *Hadoop* sistēmu ar esošo sistēmu, kā piemēram, *Oracle* konektori, *SAP HANA Vora* konektors, šāda veida risinājumi ļauj pielāgot esošās sistēmas sadarbībai ar jaunajām tehnoloģijām un neprasa lielus papildus ieguldījumus infrastruktūrā, kas bieži vien ir būtisks iemesls kāpēc jaunās tehnoloģijas netiek ieviestas.

Tiek attīstīti arī rīki, kas ļauj lietotājiem ar vienu pieprasījumu iegūt datus no vairākām sistēmām – gan relāciju datubāzes, gan *Hadoop* klastera. Tā kā lietotāji, kas strādā ar esošajām *RDBMS* sistēmām ir pieraduši pie *SQL* pieprasījumu valodas, tad ražotāji attīsta ideju lietot *SQL* valodu pieprasījumu veidošanā. *Oracle* to realizē ar *Big Data SQL* funkcionalitāti, kas tika ieviesta sākot ar datubāzes 12c versiju un ļauj piekļūt datiem no dažādiem avotiem, lietojot *SQL* valodu. *IBM* ir izveidojis *Big SQL* rīku, kas darbojas līdzīgi un ļauj ar *SQL* pieprasījumiem izgūt datus no *Hadoop* klastera. Savukārt *Teradata* liek uzsvaru uz vienotas arhitektūras izveidi, šajā arhitektūrā arī ir *Teradata QueryGrid* rīks, kas ļauj piekļūt datiem no dažādiem avotiem ar vienu *SQL* pieprasījumu.

Liela uzmanība tiek pievērsta datu vizualizācijas iespējām, kā arī apvienota sistēmas pārvaldības paneļa izveidošana, kas ļautu samazināt sistēmas administrēšanas, uzturēšanas un pārvaldes nepieciešamo laiku un resursus, kā arī ļautu daudzās sistēmas savienot un pārvaldīt kā vienotu sistēmu. *Teradata Unified Data Architecture* ir viens no piemēriem šīs idejas veiksmīgai realizācijai. Arī *Microsoft* liek uzsvaru uz vienotas infrastruktūras izveidošanu.

4. RDBMS UN HDFS SISTĒMU DATU APVIENOŠANA

Aizvien vairāk parādās nepieciešamība pēc sistēmām, kas apvieno tradicionālo *RDBMS* sistēmu un *Hadoop* funkcionalitāti, jo lielākai daļai uzņēmumu jau eksistē informācijas sistēmas, kuru pamatā ir kāda no relāciju datubāzēm. Jaunas infrastruktūras izveide prasa lielas investīcijas un ne vienmēr ir racionāla. Tajā pat laikā aizvien vairāk datu tiek glabāti *HDFS* sistēmās, kā piemēram, sensoru dati vai informācija par lietotāju klikiem sistēmās. Rodas nepieciešamība analizēt relāciju datubāžu datus un *HDFS* datus vienkopus, piemēram, analizēt lietotāju uzvedību interneta veikalā – kādas preces un preces parametrus lietotājs ir apskatījies un ko ir nopircis. Tas ļauj labāk izprast lietotāju uzvedības paradigmu. Informācija par veiktajiem pirkumiem atrodas *RDBMS* sistēmā, savukārt informācija par to, ko lietotājs ir skatījies *HDFS* sistēmā. Tādējādi ir nepieciešama sistēma, kas ļauj izveidot vienotu skatījumu uz datiem no dažādiem avotiem. 2015.gadā tika publicēts *IBM* pētījums [9], kurā tika analizēts hibrīdās datu noliktavas modelis – daļa datu atrodas relāciju datubāzē un daļa datu *HDFS* klasterī, pētījumā tika analizētas datu savienošanas iespējas un šo risinājumu veiktspēja. Sistēma tika veidota balstoties uz *IBM* piedāvātajiem produktiem – *IBM DB2* tika izmantota kā relāciju datubāze un *IBM Big SQL* risinājums, lai testētu datu savienošanu relāciju datubāzes pusē. Šajā darbā tiks realizēta un analizēta hibrīdās datu noliktavas darbība, izmantojot *Oracle* produktus, kā relāciju datubāze tiks izmantota *Oracle* Datubāzes 12c versija, datu savienojumu testēšanai datubāzes pusē tiks lietots *Oracle Big Data SQL* produkts.

4.1. Testējamās sistēmas implementācija

Testējamā sistēmas pamatā ir dators ar šādiem parametriem:

- *Windows 7 Professional 64* - bit operatīva sistēma
- *Intel(R) Core(TM) i5 – 4300U CPU @1.90GHz* 4 kodoli
- *16 GB RAM*

Pati sistēma tika izveidota, izmantojot virtualizāciju - *Oracle VM VirtualBox* versija 4.3. Izmantojamās operētājsistēmas versija ir *Oracle Enterprise Linux* 6.7. Lai izveidotu hibrīdo datu noliktavu tika uzinstalēta *Oracle RDBMS* datubāze - Oracle Database *12c Release 1 Enterprise Edition (12.1.0.2).* Tajā ir iekļauta *Oracle Big Data SQL* ārējās

tabulas, *Oracle Multitenant*, *Oracle OLAP* un *Oracle Advanced Analytics* funkcionalitātes. *Oracle* datubāze tika veidota kā *Oracle RAC* sistēma, tas pēc būtības ir *RDBMS* klasteris, kas nodrošina datubāzes augstu pieejamību, ja kaut kas notiek ar vienu klastera mezglu, otrs turpina darbību un pārņem visu datubāzes slodzi, kamēr pirmā mezgla darbība tiek atjaunota, tādējādi datubāzes procesi neapstājas. Klasteris ļauj nodrošināt arī slodzes balansēšanu un procesu paralelizēšanu. Šajā eksperimentā tika izveidota *Oracle* Datubāze ar divām instancēm – RAC1 un RAC2. Lai nodrošinātu *Oracle RAC* darbību tika uzinstalēta *Oracle Grid 12.1.0.2* programmatūra, kas nodrošina mezglu savstarpēju komunikāciju un uztur metadatus par *Oracle* klasterī esošajiem mezgliem. *Oracle Grid 12.1.0.2*. ir arī nepieciešams *Oracle Big Data SQL* rīka darbībai, kurš ir nepieciešams datu savienojumu testēšanai. [25, 34]

Implementācijā ietilpst arī *Apache Hadoop Cloudera Distribution (CDH5.5.1),* kurā ir iekļauti *HDFS, MapReduce* un *YARN,* un *Cloudera Manager (*5.5.1), kas nodrošina datu noliktavas *HDFS* daļas darbību un pārvaldi. *Hadoop* klasteris šajā eksperimentā sastāv no viena *Master* mezgla *Namenode* un trīs datu mezgliem – *Datanodes.*

RDBMS datubāzē ir optimizer komponente, kas palīdz datubāzei izvēlēties optimālo izpildes plānu katram pieprasījumam, lai patērētu pēc iespējas mazāk datubāzes resursu, datu indeksēšanas atbalsts, kas palīdz paātrināt nepieciešamo datu meklēšanu tabulās, kā arī *SQL* valodas atbalsts, kas ir nepieciešama, lai veiktu pieprasījumus – atrast un nolasīt vajadzīgos datus, kā arī veikt citas manipulācijas ar datiem (atjaunot, pievienot, dzēst). Savukārt *HDFS* pusē šajā eksperimentā datu apstrāde notiek izmantojot pilnu datu skenēšanu bez jebkādas indeksēšanas atbalsta, tā datu apstrāde notiek visās *SQL-on-Hadoop* sistēmās, kā piemēram, uz *Spark* balstītu *Shark, Impala,* uz *MapReduce* balstītu *Hive* sistēmu. Šajā eksperimentā tiks izmantots *Hive* produkts, lai veiktu datu apvienošanu *HDFS* pusē.

Jāņem vērā arī tas, ka parasti reālajās situācijās datu apjoms *RDBMS* sistēmas tabulās ir krietni mazāks nekā datu apjoms *HDFS* sistēmas tabulās.

Datu apvienošanas ātrdarbības testēšanā tika izmantoti trīs rīki – *Oracle Big Data SQL, Oracle SQL* konektors priekš *HDFS* un *Hive* rīks.

4.1.1. Hive rīka arhitektūra un konfigurācija

Hive ir datu noliktavu infrastruktūras rīks, kas ir radīts, lai apstrādātu stukturētos *Hadoop* datus. Tas tiek izmantots, lai vienkāršotu pieprasījumu veikšanu un analīzi, pieprasījumu veikšanai tiek lietota *HiveQL* (*HQL*) valoda, kas ir līdzīga *SQL* pieprasījumu valodai, lai gan ir arī savas atšķirības. [17]

Pamatā dati *Hive* sistēmā ir organizēti:

- Tabulās – sava veida analogs relāciju datubāžu tabulām. Tās ir iespējams filtrēt, apvienot ar citām tabulām, veidot projekcijas. Tiek atbalstītas arī ārējās tabulas;
- Partīcijas – katra tabula var tikt sadalīta arī sīkāk partīcijās, attiecīgi tiek veidots arī sadalījums pa direktorijām *HDFS* sistēmā;
- Dati partīcijās var tikt sadalīti arī vēl sīkākās daļās, balstoties uz kolonnas hash vērtību tabulā. Katra šāda daļa tiek attiecīgi glabāta partīcijas direktorijā *HDFS* sistēmā.

Galvenās līdzības *SQL* sintaksei ir tabulu veidošanā, datu ielādēšanā un pieprasījumu veikšanā. *HiveQL* valoda ļauj veikt datu ievietošanu arī vairākās tabulās vienlaicīgi – izmantojot vienu pieprasījumu, kā arī atbalsta iekļaut *MapReduce* kodu pieprasījumos.

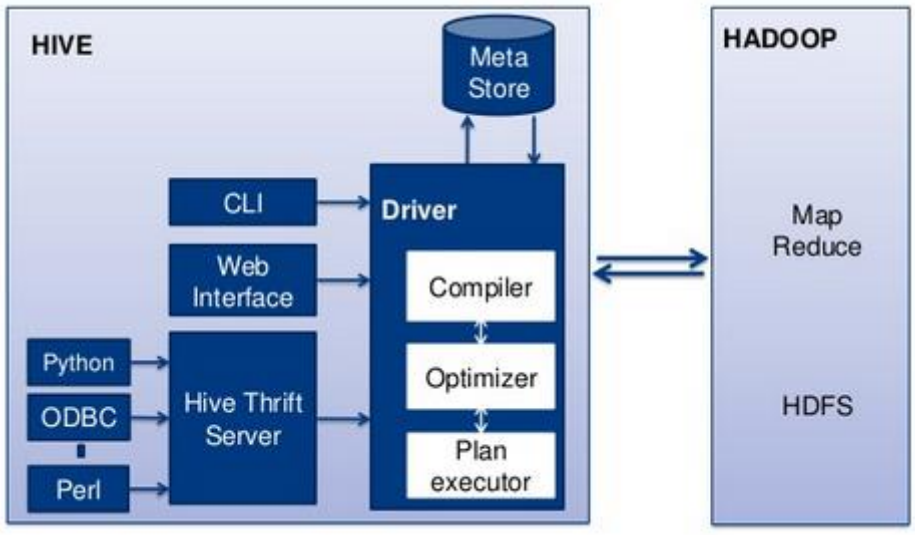
Hive arhitektūra ir attainota attēlā 4.1.1. Ar *HDFS* sistēmu *Hive* komunicē izmantojot *MapReduce* valodu, savukārt ar *RDBMS* sistēmām ir iespējams lietot *SQL* valodu. Gadījumos, kad tiek veidota datu apvienošana, tiek izmantots broadcast tipa datu savienojums, tāpēc šis rīks ir nepieciešams šī eksperimenta veikšanai – tas veiks broadcast tipa datu savienojumu *HDFS* sistēmā.

Galvenās arhitektūras sastāvdaļas ir kompilētājs, kas analizē pārbauda pieprasījumu semantiku – vai ir nepieciešams konvertēt datatipus, vai kolonnu vārdi ir uzrakstīti bez kļūdām. Pārveido pieprasījumus loģiskajā plānā, kas sastāv no relāciju algebras operatoriem un speciāliem *Hive* operatoriem, lai pēc tam šo plānu varētu pārveidot *MapReduce* uzdevumā. Kā arī ģenerē izpildes plānu. [13]

Otra svarīga arhiktūras komponente ir *Hive* optimizētājs. Tas automātiski atpazīst dažādus gadījums, kad ir nepieciešams optimizēt pieprasījumu izpildi, kā piemēram:

- Zvaigznes shēmas datu savienojumi;

- *Map* datu savienojumi ir automātiski – *MAPJOIN* savienojums nozīmē, ka datu apvienošanas laikā mazākā tabula tiks pilnībā ielādēta atmiņā un tad apvienota ar lielāko tabulu.



4.1.1.att. ***Hive*** **arhitektūra [13]**

Hive metadati glabā informāciju par *Hive* tabulām un partīcijām relāciju datubāzē, kā arī ļauj lietotājiem piekļūt šai informācijai, lietojot metadatu *API* servisu.

Hive Thrift Server ir papildus serviss, kas ļauj attālināti piekļūt *Hive* sistēmai, iesniegt pieprasījumus un izvadīt rezultātu, pieprasījumu veikšanai ir iespējams izmantot dažādas valodas, kā piemēram, *Perl* vai *Python*. [13, 17]

Lai pieslēgtos *Hive* sistēmai ir iespējams izmantot komandrindas saskarni – *CLI*, kas *Hive* gadījumā ir *$HIVE_HOME/bin/hive* programma, kas ļauj interaktīvā režīmā ievadīt dažādas komandas, lai noskaidrotu sistēmas stāvokli, kā arī veikt pieprasījumus. *$HIVE_HOME* ir sistēmas mainīgais, kas satur pilnu ceļu uz direktoriju, kurā *Hive* sistēma ir uzinstalēta. Pieslēgšanās saskarnei notiek caur komandrindas termināli, *Linux* gadījumā pieprasījuma veikšana notiek šādi:

```
$HIVE_HOME/bin/hive -e 'select a.kolonnas_nosaukums from tabulas_nosaukums a'
```

Opcija –e norāda, ka pieprasījums tiks izpildīts batch režīmā, ja –e opcija nav norādīta, tad pieprasījuma izpilde notiek interaktīvajā režīmā. Komandrindas saskarne pieprasījumu testēšanas laikā netika lietota, jo tā nav parocīga, ja ir nepieciešams veikt labojumus pieprasījumā.

Hive sistēmai ir arī grafiska lietotāju saskarne, kas arī tika lietota pieprasījumu testēšanas laikā. Tā ir intuitīvi saprotama un ļauj vienkārši modificēt esošos pieprasījumus. Kā arī palīdz atrast kļūdas pieprasījumos, ja tādas ir.

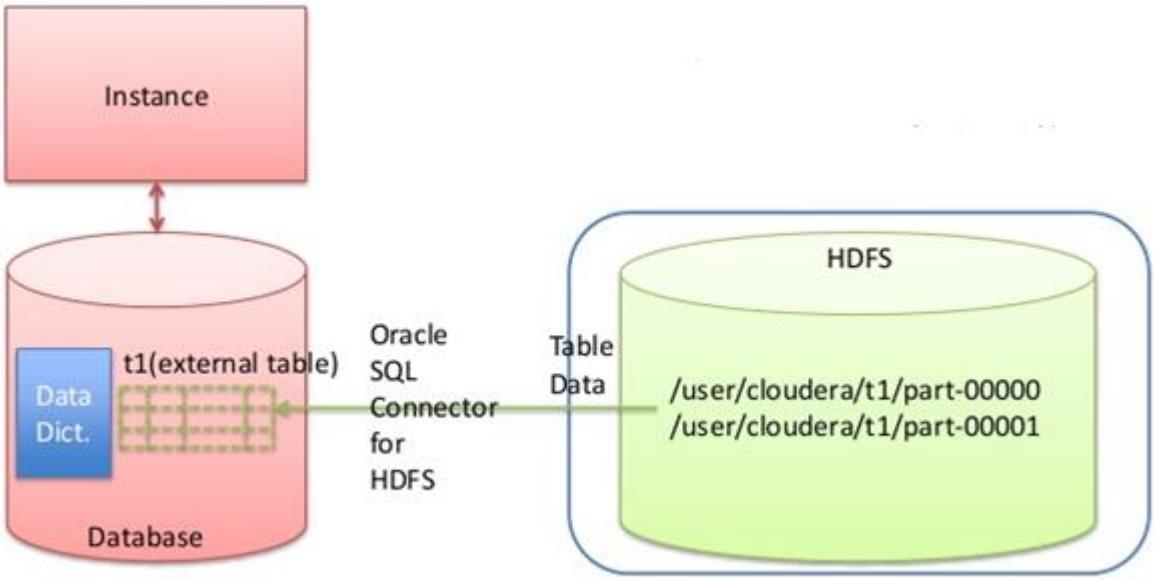
Hive instalēšana ir vienkārša, ir nepieciešama *Java 1.7.* versija un *Hadoop 2.0* vai augstāka versija. Pats instalācijas process ir paveicams ar vienu komandu – *Hive* nepieciešamie dati atrodas tar arhīvā, tāpēc izvēlētajā direktorijā vienkārši tiek izvilkti no tar arhīva visi faili. Testa sistēmā, kas ir *Linux* sistēma, tas tika izdarīts ar tar -xzvf hive-1.2.0.tar.gz komandu.

4.1.2. Oracle SQL konektora priekš HDFS arhitektūra un konfigurācija

Šī konektora lietošana ļauj *Oracle* datubāzei piekļūt un analizēt datus *HDFS* sistēmā, datu formāti, kas tiek atbalstīti ir:

- *DataPump* faili *HDFS* sistēmā – *DataPump* ir *Oracle* rīks, kas ļauj eksportēt un importēt datus – tabulas, shēmas, tabulvietas – uz dažādām datubāzēm;
- *HDFS* teksta faili, kur ir izmantoti atdalītāji;
- *Hive* tabulas.

Ja dati ir tādos formātos kā *JSON*, tad ir nepieciešams sākumā ievietot šos datus *Hive* tabulā un tikai pēc tam *Oracle SQL* konektors spēs atpazīt šos datus.



4.1.2.att. ***Oracle SQL* konektora priekš *HDFS* arhitektūra [22]**

Lai nolasītu *HDFS* datus tiek lietotas ārējās tabulas, kas identificē datu, kas ir ārpus *Oracle* Datubāzes atrašanās vietu. Attēlā 4.1.2. ir parādīts *Oracle SQL* konektora darbības princips. Ārējās tabulas uztur informāciju par *HDFS* datu atrašanās vietu. [22]

Šī rīka lietošanai ir nepieciešams papildus vides mainīgais *HADOOP_CLASSPATH*, kas iekļauj sevī ceļu uz *Oracle SQL* konektora *JAR* failu atrašanās vietu diskā.

Kā arī ir jālieto speciāls *ExternalTable* rīks, kas ļauj:

- Izveidot nepieciešamās ārējās tabulas;
- Atrast atrašanās vietu failus *HDFS* sistēmā;
- Ievietot informāciju par atrašanās vietu failiem ārējās tabulās;
- Veidot ārējās tabulas definīciju.

Atrašanās vietu faili ir faili, kuros ir *Universālie Resursu Identifikatori (URI),* kas ļauj atrast datu failus, paši datu faili atrodas *HDFS* sistēmā.

SQL konektors atbalsta tikai neparticionētās *Hive* tabulas, kuras ir definētas izmantojot *ROW FORMAT DELIMITED* un *FILE FORMAT TEXTFILE* parametrus. Ir atbalsts gan *Hive* pārvaldītām tabulām, gan *Hive* ārējās tabulām.

Lai izveidotu ārējo tabulu priekš teksta failiem ar atdalītājiem, tiek norādītas konfigurācijas īpašibas, kuras nosaka kolonnu skaitu, delimitera veidu un nepieciešamības gadījumā ārējās tabulas kolonnu nosaukumiem. Šī eksperimenta laikā tiek lietotas ārējās tabulas priekš teksta failiem ar atdalītājiem, jo ģenerētie dati ir *csv* formātā.

Ārējās tabulas izveidošanai sākumā ir nepieciešams izveidot direktoriju *RDBMS* sistēmas serverī, to var izdarīt ar komandu:

```
mkdir /u01/oradata/RAC/external
```

Pēc tam tiem izveidota direktorija *Oracle* datubāzē, lai to būtu iespējams lietot, tad tiek piešķirtas lasīšanas un rakstīšanas tiesības datubāzes lietotājam, kurš veiks pieprasījumus.

```
$ sqlplus / as sysdba
SQL> CREATE OR REPLACE DIRECTORY store_sales_dir AS
'/u01/oradata/RAC/external '
SQL> GRANT READ, WRITE ON DIRECTORY store_sales_dir TO hadoop;
```

Tiek definēti sistēmas mainīgie *OSCH_HOME* un *HADOOP_CLASSPATH*, kas norāda uz vajadzīgo bināro failu atrašanās vietām.

```
$ export OSCH_HOME="/opt/oracle/orahdfs-2.0"
$ export HADOOP_CLASSPATH="$OSCH_HOME/jlib/*:$HADOOP_CLASSPATH"
```

Un tiek izveidota ārējā tabula

```
$ hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
-D oracle.hadoop.exttab.tableName=STORE_SALES \
-D oracle.hadoop.exttab.locationFileCount=2 \
-D
oracle.hadoop.exttab.dataPaths="hdfs:///data/s1/*.csv,/data/s2/*.c
sv " \
-D oracle.hadoop.exttab.columnCount=23 \
-D oracle.hadoop.exttab.defaultDirectory=STORE_SALES_DIR \
-D
oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/RAC \
-D oracle.hadoop.connection.user=hadoop \
-createTable
```

Ir iespējams izveidot arī konfigurācijas failu *XML* formātā, kas saturēs visu nepieciešamo informāciju par ārējo tabulu parametriem. Izmantotā konfigurācijas faila saturs ir pievienots 4.pielikumā.

4.1.3. Oracle Big Data SQL arhitektūra un konfigurācija

Oracle Big Data SQL atbalsta pieprasījumus uz citām sistēmām, tādām kā *Apache Hive, HDFS, Apache Hbase* un *Oracle NoSQL* datubāzi. Viss tiek veikts ar vienotu *SQL* pieprasījumu un nav nepieciešamības to pielāgot katrai sistēmai atsevišķi.

Sākot ar *Oracle* Datubāzes 12.1.0.2. versiju tiek atbalstīti divi jauni piekļuves draiveri priekš *Big Data SQL*, kas optimizēt pieprasījumu veiktspēju:

- *ORACLE_HIVE* – ļauj veidot *Oracle* ārējās tabulas kā datu avotus no *Apache Hive*.
- *ORACLE_HDFS* – ļauj veidot *Oracle* ārējās tabulas pa tiešo *HDFS* sistēmā, kā *HDFS* failus

Tā kā ārējās tabulas nesatur tradicionālos indeksus kā tas ir *Oracle* relāciju datubāzē, tad pieprasījumu veikšanai ir nepieciešams lietot pilnu tabulas skenēšanu, lai atrastu vajadzīgos datus. *Oracle Big Data SQL* šim nolūkam izmanto *SmartScan* tehnoloģiju, tas ļauj atlasīt vajadzīgos datus un rezultējošās datu kopas ir krietni mazākas, līdz ar to samazinās arī tīkla noslodze un laiks, kas nepieciešams datu pārsūtīšanai uz *RDBMS*.

Oracle Big SQL arī automātiski uztur disku indeksus, kas glabā informāciju kopsavilkuma veidā par *HDFS* datu sadalījumu cietajos diskos. Šāda veida informācija ļauj samazināt *I/O* darbību noslodzi un *CPU* noslodzi, jo parastie faili tiek mazāk konvertēti *Oracle* Datubāzes blokos. Uz doto brīdi šī funkcionalitāte attiecas tikai uz ārējām tabulām, kas ir balstītas uz *HDFS* un ir veidotas lietojot *ORACLE_HDFS* draiveri vai *ORACLE_HIVE* draiveri. Tā kā disku indekss glabā minimālās un maksimālās kolonnu vērtības, tad notiek arī tā saucamā *I/O* filtrēšana, jo nav nepieciešamas *I/O* operācijas, jo ir zināms, ka konkrētie bloki nesatur vajadzīgo informāciju. [25]

Oracle Big Data SQL instalēšanai ir nepieciešami daudzi prerekvizīti, jo bez tiem *Big Data SQL* nevarēs strādāt. Rīki, kuriem ir jābūt uz servera:

- *HDP 2.3 – Hortonwork Data Platform*, kas ir atvērtā koda *Apache Hadoop* distribūcija, kura ir balstīta uz *YARN* arhitektūru;
- *Ambari 2.1.0* – atvērtā koda *Apache Hadoop* klasteru pārvaldības platforma, kas atvieglo darbu ar *Hadoop* – palīdz monitorēt klastera stāvokli, veidot jaunus mezglus un nodrošina drošības prasības; [20]
- *HDFS 2.7.1- Apache Hadoop* sadalītā failu sistēma;
- *YARN 2.7.1 – Hadoop* klastera resursu pārvaldes platforma, nodrošina dažādu datu apstrādes rīku darbību, datu plūsmu reālajā laikā. Arhitektūrā darbojas kā centrālā platforma, tāpēc tiek saukta arī par datu operētājsistēmu; [16]
- *Zookeeper 3.4.6* – rīks, kas palīdz pārvaldīt *Hadoop* servera konfigurācijas, nosaukumu veidošanu, sinhronizāciju starp dažādām sistēmas daļām.
- *Hive 1.2.0* – rīks, kas ļauj veidot *Hive QL* pieprasījumus;
- *Tez 0.7.0 – Hadoop* datu apstrādes programma, kas uzlabo *MapReduce* darbības paradigmu, ļaujot uzdevumus, kuru risināšanai pirms tam bija vajadzīgas vairākas *MapReduce* operācijas apvienot vienotā *Tez* uzdevumā. [16]

Kā arī jābūt:

- *JDK* versija 1.7 vai lielāka
- *Python* versija 2.6.
- *OpenSSL* versija 1.01 build 16 vai lielāka
- *Oracle Instant Client* – 12.1.0.2 vai lielāka
- *Oracle Instant JDBC Client* – 12.1.0.2 vai lielāka
- *Apache log4j*

Eksperimenta laikā tika lietota *Oracle Big Data Light* virtuālā mašīna 4.4.0 versija, kurā prerekvizītu rīki jau bija uzinstalēti. [15]

Savukārt paša *Oracle Big Data SQL* rīka instalēšana ir ļoti vienkārša un neprasa daudz laika. Lai lietotu rīku ar *HDFS* vai *Hive* tabulām, tad izmantojot *ORACLE_HDFS* vai attiecīgi *ORACLE_HIVE* draiverus ir jāveido ārējās tabulas pa tiešo *HDFS* vai *Hive* sistēmās. Eksperimenta vajadzībām tika izveidota ārējā tabula STORE_SALES, tās izveidošanai tika izmantota *ORACLE_HDFS* funkcionalitāte, jo tādā gadījumā *Oracle* datubāze neizmanto *Hive* sistēmas tabulu metadatus un var notestēt, kā notiek datu apvienošana, ja nav pieejama papildus informācija par datiem. Ārējās tabulas izveidošanai tika izmantota komanda:

```
CREATE TABLE store_sales    (ss_sold_date_sk VARCHAR2(10),
                             ................
                             ss_net_profit NUMBER(7,2) )

        ORGANIZATION EXTERNAL
        (
          TYPE oracle_hdfs
          DEFAULT DEFAULT_DIR

      LOCATION ('hdfs:/data/s1/*.csv'));
```

Pilns komandas teksts atrodas 5.pielikumā. Tā tika saīsināta garā kolonnu saraksta dēļ, piemērā ir attēlotas tikai pirmā un pēdējā tabulas kolonna. Tabulas definīcijā tiek norādīts tabulas organizācijas veids – *External*, kas nozīmē, ka tabula ir ārējā, kā arī *HDFS* failu atrašanās vietas *Hadoop* klasterī. [25]

4.2. Datu apvienošanas veidi

Lai apvienotu datus no dažādiem datu avotiem, kā piemēram, *HDFS* un *RDBMS* var izmanto *SQL* pieprasījumus. Pieņemsim, ka transakciju datubāzes tabulā P glabājas dati par pārdotajām precēm un *HDFS* pusē tabulā T glabājas dati par precēm, kuras tika apskatītas interneta veikalā. *SQL* pieprasījums, kas parādītu, kādus produktus lietotāji no Somijas, kuri ir nopirkuši Samsung televīzoru, ir apskatījušies, var izskatīties šādi:

```
SELECT T.web_page, COUNT(*)
FROM P, T
WHERE P.product = ‘Samsung TV’
AND country(T.ip)= ‘Finland’
And P.uid=T.uid
AND P.pdate >= T.tdate AND P.pdate <= T.tdate+1
GROUP BY T.web_page
```

Šajā pieprasījumā tiek meklēts, cik reizes un kādus produktus apskatīja lietāji no valsts Somija (tiek atrasts pēc *IP* adreses, tāpēc rezultāts var nebūt precīzs, ja lietotājs no citas valsts ir pieslēdzies tīklam Somijā, piemēram, darba *VPN*) un kuri ir nopirkuši Samsung televīzoru. Pieprasījums atrod tikai vienas dienas laikā apskatītos un nopirktos produktus. Pieprasījums, kas tiek izmantots piemērā, satur sevī gan lokālos predikātus, gan predikātus apvienojuma rezultātā, gan agregācijas funkciju.

Lai veiktu datu apvienošanu vajadzīgie dati no vienas sistēmas tiek pārsūtīti uz otru, tāpēc ir nepieciešams minimizēt pārsūtīto datu apjomu starp sistēmām. Paša *SQL* pieprasījuma pārsūtīšana, kā arī gala rezultātu atgriešana no *HDFS* pieprasa relatīvi maz resursus. Taču pašu datu pārvietošana var patērēt daudz resursu, īpaši, ja tabula ir liela un selektivitāte ir maza. Lai samazinātu pārvietoto datu apjomu, kas nepieciešams datu apvienošanai, tiek izmantoti Blūma filtri (*Bloom filters*). [7, 9]

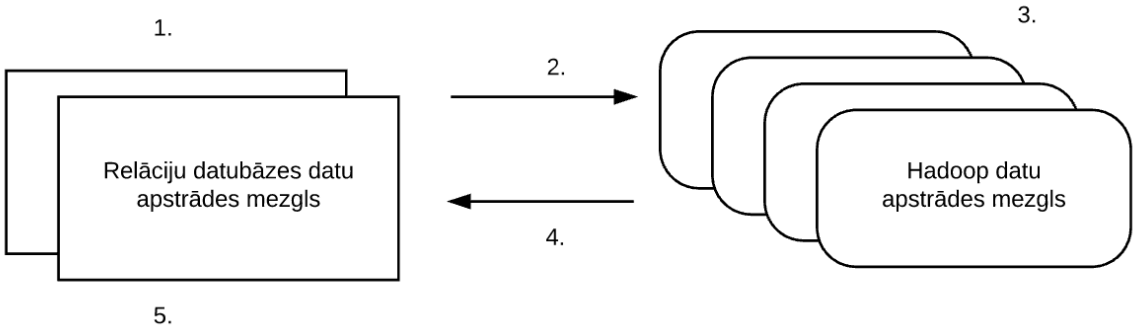
Blūma filtrs pēc būtības ir m bitu masīvs ar k hash funkcijām, kas apvieno elementu kopu, to lieto, lai noskaidrotu vai elements pieder kopai. Ja Blūma filtram tiek pievienots papildus elements, tam tiek piemērotas k hash funkcijas un attiecīgi izmainīti atbilstošie biti masīvā uz vērtību 1. Blūma filtrs ir parocīga un efektīva datu struktūru, lai testētu savienojumu selektivitāti. [7]

4.2.1. Datu apvienotšana RDBMS datubāzes pusē

Tradicionālā *Oracle* un arī daudzu citu ražotāju, kā piemēram, *TeraData SQL-H*, *Microsoft Polybase*, pieeja ir atrast vajadzīgo tabulu *HDFS* sistēmā un izpildīt datu apvienošanu *RDBMS* datubāzes pusē. *Oracle* šim nolūkam izmanto *Oracle Big Data SQL* produktu, kas ļauj ar vienu *SQL* pieprasījumu nolasīt datus no dažādām sistēmām – gan relāciju datubāzēm, gan *HDFS*, gan *NoSQL* datubāzēm.

Vienkāršākajā gadījumā *HDFS* pusē datiem tiek piemēroti projekcijas un lokālie predikāti un tad šie dati tiek nosūtīti uz *RDBMS* datubāzi. Šajā gadījumā veiktspēju ietekmē datu apjoms, kas no *HDFS* ir jāpārsūta, to ietekmē projicējamo kolonnu izmērs un predikātu selektivitāte. Ir jāņem vērā, ka *HDFS* datu apjoms parasti ir ievērojami lielāks par *RDBMS* datu apjomu, tāpēc arī neskatoties uz to, ka dati tiek atlasīti balstoties uz predikātiem, gala rezultātu apjoms var būt liels. Tāpēc tiek izmantoti Blūma filtri, kas palīdz samazināt pārsūtīto datu apjomu.

Blūma filtra darbības mehānisms datu apvienošanas gadījumā ir vienkāršs, tas ir shematiski parādīts attēlā 4.2.1.



4.2.1.att. **Datu apvienošana *RDBMS* pusē [9]**

Pamatdarbības, kas notiek datu apvienošanas laikā ir:

1. *RDBMS* datubāzes katrā no mezgliem tiek izskaitļots Blūma filtrs lokālai partīcijai un pēc tam tiek izpildīta agregācija, lai izveidotu globālo Blūma filtru;
2. Globālais Blūma filtrs tiek nosūtīts uz *HDFS* sistēmu;
3. *HDFS* sistēmā tiek noskenētas vajadzīgā tabula un tiek piemēroti lokālie predikāti, pēc predikātu piemērošanas tiek pielietots atsūtītais Blūma filtrs;
4. Iegūtie rezultāti tiek nosūtīti atpakaļ uz *RDBMS* sistēmu;
5. Tiek izpildīta nepieciešamā datu apvienošana un agregācija.

Datu apvienošanas testēšanai *RDBMS* pusē tiks lietoti *Oracle Big Data SQL* rīks, kas savā darbībā izmanto Blūma filtrus un *Oracle SQL* konektors priekš *HDFS,* kas veic datu apvienošanu *RDBMS* pusē, bet pārsūtāmo datu apjoma samazināšanai neizmanto Blūma filtrus. [33, 25]

4.2.2. Datu apvienotšana HDFS sistēmas pusē – broadcast join

Šis datu apvienošanas veids ir efektīvs tad, ja *RDBMS* pusē tabulas predikāti ir ļoti selektīvi, tādējādi mazs datu apjoms tiek sūtīts uz *HDFS* sistēmu. Relāciju datubāzes dati tiek nosūtīti uz katru no *Hadoop* datu apstrādes mezglu, tas nozīmē, ka var tikt pielietota tikai lokāla datu apvienošana, bez datu sajaukšanas *HDFS* pusē. Šī datu apvienošanas algoritma gadījumā arī agregācija notiek *HDFS* sistēmā, tāpēc atpakaļ uz *RDBMS* sistēmu tiek nosūtīti mazs rezultējošo datu apjoms.

Šīs metodes darbības mehānisms:

1. Katrā datubāzes mezglā tiek piemēroti lokālie predikāti un projekcijas;
2. Relāciju datubāzes dati tiek nosūtīti uz katru *Hadoop* datu apstrādes mezglu;
3. Tiek noskenēta vajadzīgā *HDFS* tabula un pielietoti lokālie predikāti un projekcijas, kā arī notiek datu apvienošana;
4. Tiek veiktas nepieciešamās rezultējošo datu agregācijas;

Lai realizētu šo datu apvienošanas veidu testu laikā tika izmantots *Apache Hive* produkts. [5, 9]

4.3. Datu savienojumu testēšana

Eksperimenta laikā tika testēti 3 datu apvienošanas gadījumi:

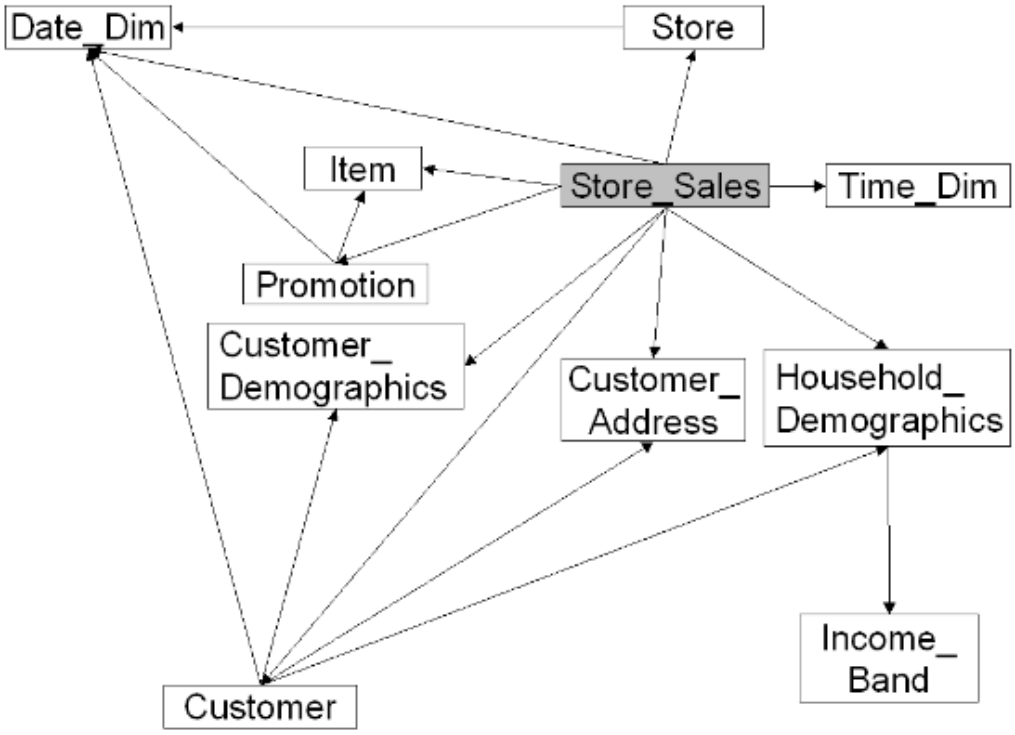
1. Datu apvienošana *RDBMS* pusē, izmantojot *Oracle Big Data SQL* produktu;
2. Datu apvienošana *RDBMS* pusē izmantojot, *Oracle SQL* connector priekš *HDFS*;
3. Datu apvienošana *HDFS* pusē, izmantojot *Hive* produktu.

Lai arī datu apvienošana notiek *RDBMS* pusē *Oracle Big Data SQL* un *Oracle SQL* konektoram priekš *HDFS* ir dažāds darbības princips. *Oracle Big Data SQL* datu apvienošanas laikā pa tiešo piekļūst *HDFS* datiem un izpilda vajadzīgās darbības, savukārt *Oracle SQL* konektors priekš *HDFS* izmanto ārējās tabulas datu apvienošanai. Sākumā vajadzīgie dati no *HDFS* tiek nosūtīti uz *Oracle RDBMS* ārējo tabulu un datu apvienošana notiek starp *RDBMS* tabulu un ārējo tabulu. *Apache Hive* produkts palīdz realizēt vienkāršo *broadcast* datu apvienošanas gadījumu, lai pieslēgtos *RDBMS* datubāzei un dabūtu no tās vajadzīgos datus tiek izmantoti *JDBC/ODBC* draiveri, bet visa datu apstrāde notiek *HDFS* pusē.

Eksperimentālie dati tika ģenerēti izmantojot *TCP-DS Benchmark* [27], kas ļauj ģenerēt un pētīt liela apjoma datus lēmumu pieņemšanas sistēmās (*DSS - Decision Support Systems*), tas ir paredzēts izmantošanai *Big Data* risinājumos – gan *RDBMS*, gan *Hadoop/Spark* balstītās sistēmās. Kā arī sistēmu pētīšanai, kurām ir raksturīga liela *CPU* un *I/O* noslodze.

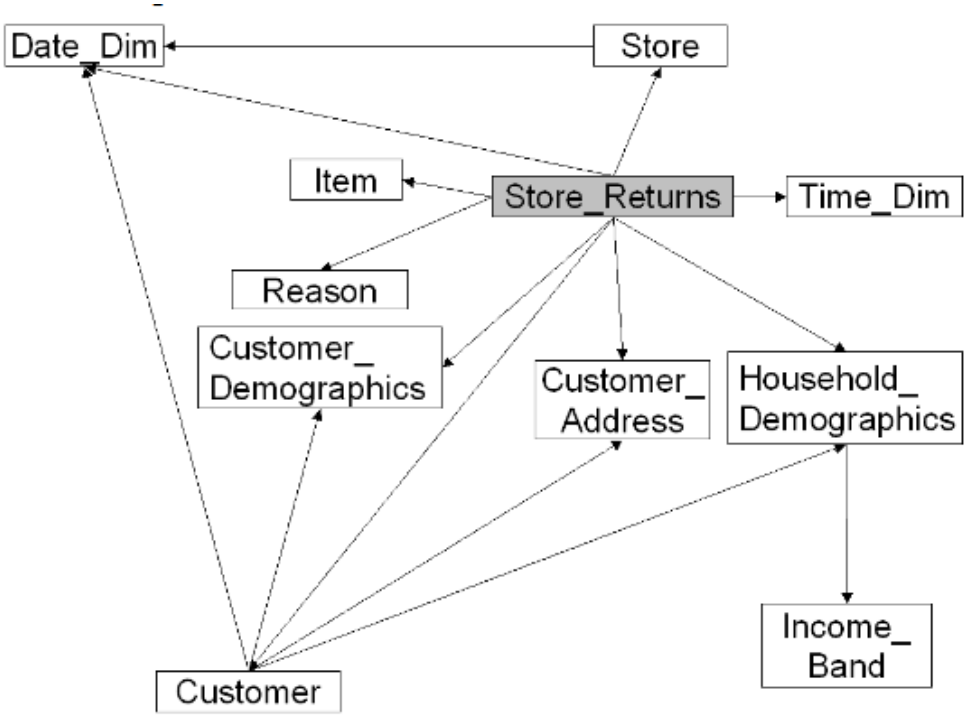
Izmantotās shēmas apraksta uzņēmuma pārdotās preces un atgrieztās preces, preču pārdošanai tiek izmantoti trīs varianti – preču katalogs, interneta veikals un veikali. Datu apvienošanas testu laikā galvenokārt tika izmantotas divas shēmas:

1. Veikala pārdotās preces – STORE_SALES, šīs shēmas *ER* diagramma ir parādīta attēlā 4.3.1. Šīs tabulas pilnā definīcija ir pievienota 1.pielikumā



4.3.att. **STORE_SALES shēmas *ER* diagramma [28]**

2. Veikala atgrieztās preces – STORE_RETURNS, šīs shēmas *ER* diagramma ir parādīta attēlā 4.3.2. Šīs tabulas pilnā definīcija ir pievienota 2.pielikumā



4.3.att. **STORE_RETURNS shēmas *ER* diagramma [27]**

Datu ģenerēšanai tika izmantots 1TB mērogs, kas nozīmē, ka apmēram tāds datu daudzums tika ģenerēts. Tiek rakstīts apmēram nevis precīzi 1TB datu apjoms, jo tas var nedaudz atšķirties atkarībā no operētājsistēmas, kurā tiek izmantoti testa skripti. Šī eksperimenta gadījumā pēc ģenerēšanas skriptu palaišanas tika radīts nedaudz lielāks nekā 1TB datu daudzums. To var pārbaudīt izmantojot *bash* skriptu komandu du –sh /tmp/dati, kur /tmp/dati ir ceļš uz ģenerēto datu atrašanās vietu.

```
[oracle@bigdata tmp] du -sh /tmp/dati
1002G           /tmp/dati
```

Visas radītās tabulas atrodas *csv* formātā, tāpēc tās ir viegli eksportēt vajadzīgajās datubāzēs.

Atbilstoši specifikācijai [27] pie 1TB liela mēroga rindiņu skaits izmantotajās tabulās STORE_SALES un STORE_RETURNS atšķiras apmēram 10 reizes. STORE_RETURNS tabulas ierakstu skaits attiecīgi ir desmit reizes mazāks nekā STORE_SALES tabulas ierakstu skaits. Tāpēc STORE SALES shēma tika izvietota *HDFS* sistēmā, savukārt STORE_RETURNS shēma tika eksportēta uz *Oracle RDBMS* sistēmu. Tabulā 4.2. ir apkopots iegūtais rindiņu skaits pēc datu ģenerēšanas. Izmantojot šos datus var izrēķināt, ka STORE_RETURNS tabulas ierakstu skaits ir 10% no STORE_SALES tabulas ierakstu skaita, kas arī bija nepieciešams testu veikšanai.

Tabula.4.1

Ģenerēto datu apjoms testa tabulās

Tabulas nosaukums	Rindiņu skaits
STORE_SALES	2 879 987 996
STORE_RETURNS	287 999 767

Datu apvienošanas gadījumā ir svarīga arī ierakstu selektivitāte. Tāpēc testi tika veikti izmantojot gan kolonnas ar augstu augstu selektivitāti, piemēram, STORE_SALES tabulā šāda kolonna ir ss_item_sk (1), kas šajā tabulā ir izmantota kā primārā atslēga – tas nozīmē, ka tās vērtības ir unikālas. STORES_RETURNS tabulas gadījumā sr_item_sk (1) kolonna ir primārā atslēga, kas *Oracle* datubāzes gadījumā nozīmē arī to, ka visas vērtības ir automātiski indeksētas. Kolonnas ar zemu selektivitāti ir ss_quantity STORE_SALES tabulā – parāda nopirkto preču skaitu, un sr_return_quantity STORES_RETURNS tabulā, kas parāda atgriezto preču skaitu. STORES_RETURNS tabulas gadījumā šīs kolonnas visbiežāk sastopamā vērtība ir 1.

Tika testēti 9 datu apvienošanas gadījumi:

1. Datu apvienošana, izmantojot predikātu ar augstu selektivitāti *HDFS* tabulā;
2. Datu apvienošana, izmantojot predikātu ar augstu selektivitāti *RDBMS* tabulā;
3. Datu apvienošana, izmantojot predikātu ar zemu selektivitāti *HDFS* tabulā;
4. Datu apvienošana, izmantojot predikātu ar zemu selektivitāti *RDBMS* tabulā;
5. Datu apvienošana un tabulas agregācija;
6. Datu apvienošana, izmantojot divus predikātus ar zemu selektivitāti, viens uz *HDFS* tabulas un otrs uz *RDBMS* tabulas, kā arī rezultātu agregācija;
7. Datu apvienošana, izmantojot divus predikātus ar augstu selektivitāti, viens uz *HDFS* tabulas un otrs uz *RDBMS* tabulas, kā arī rezultātu agregācija;
8. Datu apvienošana, izmantojot divus predikātus, viens uz *HDFS* tabulas ar augstu selektivitāti un otrs uz *RDBMS* tabulas ar zemu selektivitāti, kā arī rezultātu agregācija;
9. Datu apvienošana, izmantojot divus predikātus, viens uz *HDFS* tabulas ar zemu selektivitāti un otrs uz *RDBMS* tabulas ar augstu selektivitāti, kā arī rezultātu agregācija;

Datu apvienošanai izmantotie *SQL* pieprasījumu piemēri ir:

1. Pieprasījums, kas atrod visas atgrieztās preces, kuru identifikators ir ID205, šāds pieprasījums ļauj notestēt datu apvienošanu, ja tiek izmantots predikāts ar augstu selektivitāti uz *HDFS* tabulas.

```
SELECT *
FROM store_sales S, store_returns R
WHERE R.sr_item_sk=S.ss_item_sk
AND S.ss_item_sk=ID205;
```

2. Pieprasījums, kas atrod visas atgrieztās preces, kuru identifikators ir ID311, pieprasījums testē datu apvienošanu, ja tiek izmantots predikāts ar augstu selektivitāti uz *RDBMS* tabulas.

```
SELECT *
FROM store_sales S, store_returns R
WHERE R.sr_item_sk=S.ss_item_sk
AND R.sr_item_sk=ID311;
```

3. Pieprasījums, kas atrod visas atgrieztās preces, kuru pārdošanas cena bija mazāka par 10.00 dolāriem. Šis pieprasījums izmanto predikātu, kas ir *HDFS* pusē un tam nav augsta selektivitāte, kolonna *HDFS* tabulā nav indeksēta.

```
SELECT *
FROM store_sales S, store_returns R
WHERE R.sr_item_sk=S.ss_item_sk
AND S.ss_sales_price <= 10.00;
```

4. Pieprasījums, kas izvada informāciju par pircēju, preces pārdošanas datumu un atgriešanas datumu, ja tika atgriezts vairāk par vienu preci. Šajā pieprasījumā kā predikāts tiek izmantota kolonna ar zemu selektivitāti, tā nav indeksēta un atrodas *RDBMS* pusē.

```
SELECT S.ss_customer_sk, S.ss_sold_date_sk, R.sr_returned_date_sk
FROM store_sales S, store_returns R
WHERE R.sr_store_sk=S.ss_store_sk
AND R.sr_quantity >=  1;
```

5. Pieprasījums ar agregāciju, saskaita, cik daudz atgrieztās preces ir katrā no veikaliem. Šis pieprasījums testē gadījumu, kad tiek izmantota ne tikai datu apvienošana, bet arī agregācija. Agregācija tiek veikta tabulai, kas atrodas *RDBMS* sistēmā.

```
SELECT R.sr_store_sk, count(*)
FROM store_sales S, store_returns R
WHERE R.sr_store_sk=S.ss_store_sk
GROUP BY R.sr_store_sk;
```

6. Pieprasījums, kas atrod visus atgriezto preču skaitu pa veikaliem, ja atgrieztās preces skaits ir bijis 1 un tās pārdošanas cena mazāka par 10.00 dolāriem. Šis pieprasījums ļauj testēt gadījumus, kad tiek izmantota gan agregācija, gan datu filtrēšana ar predikātiem. Abi predikāti ir ar zemu selektivitāti, viens no tiem ir *RDBMS* sistēmā – R.sr_quantity kolonna, bet otrs *HDFS* sistēmā – S.ss_sales_price kolonna.

```
SELECT R.sr_store_sk, count(*)
FROM store_sales S, store_returns R
WHERE R.sr_store_sk=S.ss_store_sk
AND R.sr_quantity=1
AND S.ss_sales_price <= 10.00
GROUP BY R.sr_store_sk;
```

7. Šis pieprasījums ļauj testēt gadījumus, kad tiek izmantota gan agregācija, gan datu filtrēšana ar predikātiem. Abi predikāti ir ar augstu selektivitāti, viens no tiem ir *RDBMS* sistēmā – R.sr_item_sk kolonna, savukārt otrs *HDFS* sistēmā – S.ss_item_sk kolonna.

```
SELECT R.sr_store_sk, count(*)
FROM store_sales S, store_returns R
WHERE R.sr_store_sk=S.ss_store_sk
AND( R.sr_item_sk=ID222
OR S.ss_item_sk=ID444)
GROUP BY R.sr_store_sk;
```

8. Pieprasījums, kas testē agregācijas savienojumu ar diviem predikātiem, viens ir uz *HDFS* tabulas ar augstu selektivitāti un otrs uz *RDBMS* tabulas ar zemu selektivitāti

```
SELECT R.sr_store_sk, count(*)
FROM store_sales S, store_returns R
WHERE R.sr_store_sk=S.ss_store_sk
AND R.sr_quantity = 1
AND S.ss_item_sk=ID444
```

GROUP BY R.sr_store_sk;

9. Pieprasījums, kas testē agregācijas savienojumu ar diviem predikātiem, viens ir uz *HDFS* tabulas ar zemu selektivitāti un otrs uz *RDBMS* tabulas ar augstu selektivitāti

```
SELECT R.sr_store_sk, count(*)
FROM store_sales S, store_returns R
WHERE R.sr_store_sk=S.ss_store_sk
AND R.sr_item_sk=ID222
AND S.ss_sales_price <= 10.00
GROUP BY R.sr_store_sk;
```

Pieprasījumu piemēri ir izveidoti izmantojot *Oracle SQL* semantiku, tā tika lietota veicot pieprasījumu testēšanu no *Oracle* datubāzes puses, testējot *Oracle Big Data SQL* rīku un *Oracle SQL* konektoru priekš *HDFS*. Testējot *Hive* rīku šie pieprasījumi tika modificēti uz *Hive QL* valodu, taču semantikas atšķirības šajos pieprasījumos ir nebūtiskas, jo *Hive QL* valodā pieprasījumu notiek tāpat kā *SQL* valodā [18*]*. *Hive QL* galvenā atšķirība šajos pieprasījumos no *SQL* semantikas ir pieprasījumā *FROM* daļā. Veicot datu apvienošanu starp dažādām datubāzēm, tiek norādīts gan tabulas nosaukums, gan datubāzes nosaukums. Šī eksperimenta ietvaros *FROM* daļa visos pieprasījumos *Hive QL* valodā izskatījās šādi:

```
FROM hdfs.store_sales S JOIN oracleDB.store_returns R
ON R.sr_store_sk=S.ss_store_sk
```

Respektīvi *Hive QL* valodā *FROM* daļā ir jānorāda arī datubāzes nosaukums, kurā atrodas attiecīgā tabula, tas tiek darīts formātā:

```
FROM datubāzes_nosaukums.tabulas_nosaukums apzīmējums1
JOIN datubāzes_nosaukums.tabulas_nosaukums apzīmējums2
ON apzīmējums1.kolonnas_nosaukums =
apzīmējums2.kolonnas_nosaukums
```

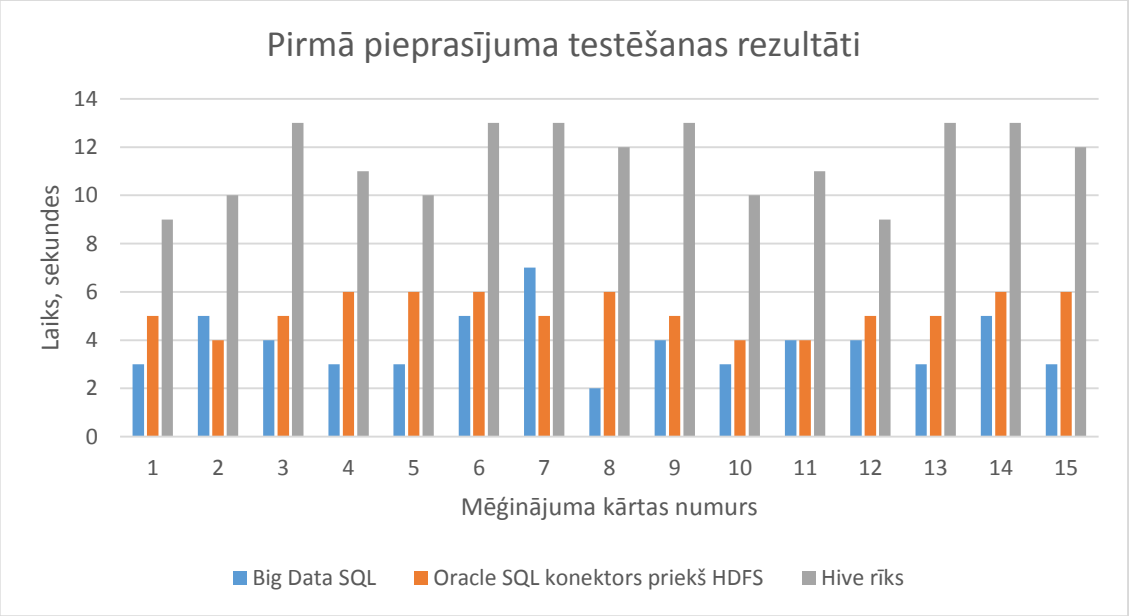
Datu apvienošanai tiek izmantota *JOIN… ON* konstrukcija, tā var tikt lietota arī *Oracle SQL* semantikas gadījumā. Pēc *ON* daļas tiek norādītas kolonnas, pēc kurām attiecīgajā pieprasījumā notiek datu apvienošana. Tā kā pieprasījumu *FROM* daļas apzīmējumu gan *Oracle SQL* semantikas, gan *Hive QL* semantikas gadījumā tika lietoti

vienādi, tad *Hive QL* gadījumā *ON* daļas pieraksts ir tieši tāds pats kā *Oracle SQL* pieraksts –apzīmējums1.kolonnas_nosaukums=apzīmējums2.kolonnas nosaukums.

4.4. Iegūtie rezultāti un diskusija

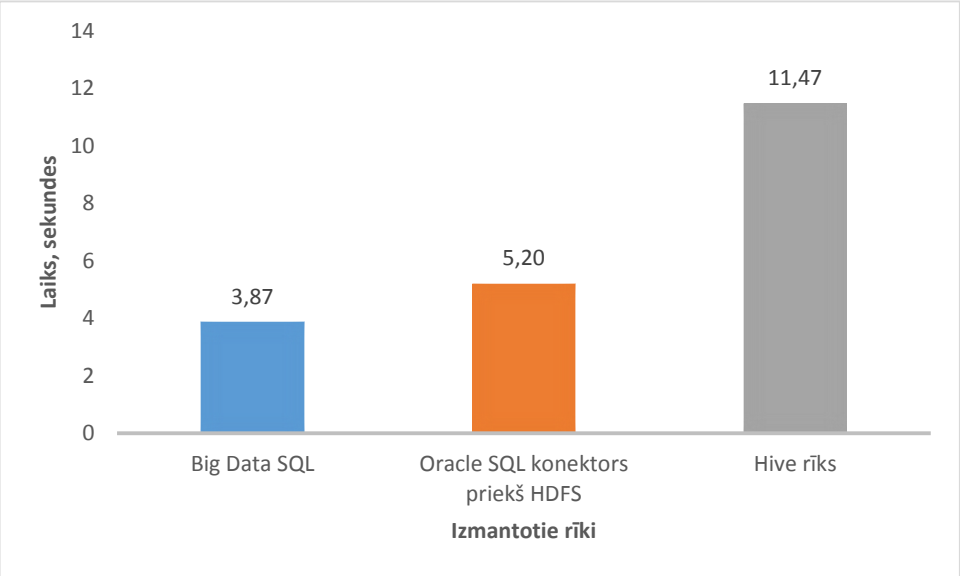
Testēšanas laikā visi pieprasījumi tika laisti 15 reizes ar katru no izvēlētajiem rīkiem rīkiem, tika mērīts pieprasījuma izpildes laiks katrā no gadījumiem. Iegūtie rezultāti ir apkopoti tabulās un attiecīgi attēloti grafiski. Tabulas ar iegūtiem rezultātiem ir atrodamas 3.pielikumā.

Pirmā pieprasījuma rezultāti ir apkopoti attēlā 4.4.1, pieprasījumā dati tika apvienoti pēc preces *ID*, kas ir abās tabulās ir kolonna ar augstu selektivitāti, kā arī šī kolonna ir indeksēta *Oracle* datubāzē. Šī kolonna tika izmantota arī kā predikāts pieprasījumā. No rezultātiem var redzēt, ka vislabāk ar šo uzdevumu tiek galā *Oracle Big Data SQL* produkts, tas pa tiešo komunicē ar *HDFS*, neveidojot ārējās tabulas kā tas ir *Oracle SQL* konektora gadījumā, tāpēc arī *Oracle SQL* konektrora rezultāts ir nedaudz sliktāks. *Broadcast* datu apvienošana ar *Hive* rīku parāda viszemāko ātrdarbību. Salīdzinot testa rezultātā iegūtās vidējās vērtības, kas ir atainotas attēlā 4.4.2. var redzēt, ka *Hive* rīks šajā gadījumā darbojas gandrīz 3 reizes lēnāk nekā *Oracle Big Data SQL* un gandrīz divas reizes lēnāk nekā *Oracle SQL* konektors priekš *HDFS*. Starp *Oracle Big Data SQL* un *Oracle SQL* konektoru priekš *HDFS* atšķirība nav tik liela, apmēram 34% lēnāks ir *SQL* konektors.



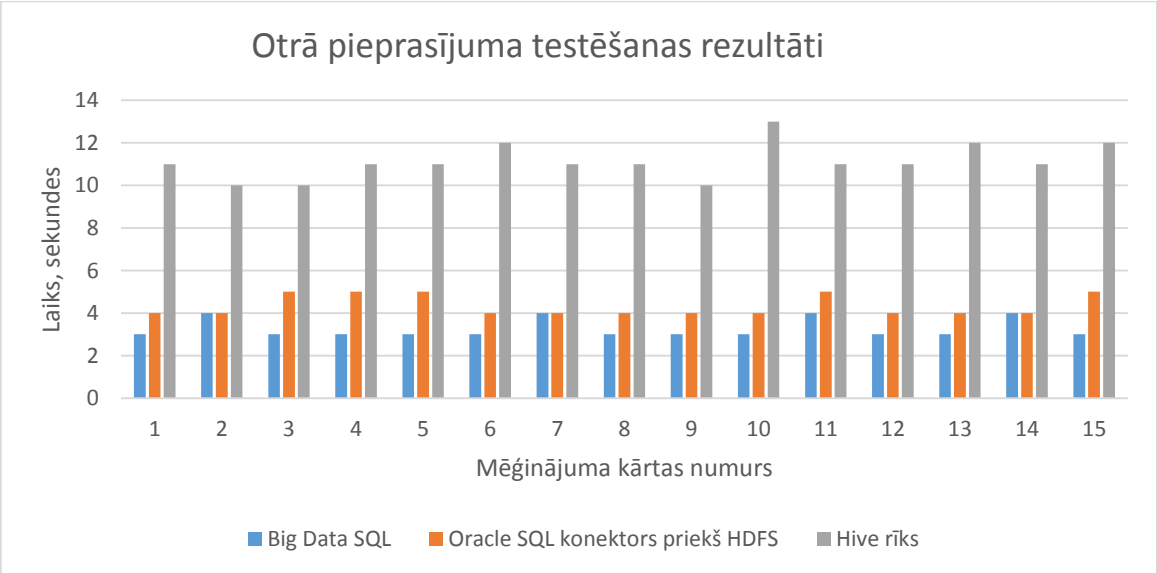
4.4.1.att. **Pirmā pieprasījuma testēšanas rezultāti**

Kaut gan, ja aplūko 4.4.1. attēlā parādītos atsevišķos rezultātus var redzēt, ka ir bijuši mēģinājumi, kad *SQL* konektors ir darbojies ātrāk, kā piemēram, mēģinājums 2 un mēģinājums 7 , kā arī, kad rezultāti ir bijuši vienādi – mēģinājums 11. To var skaidrot ar to, ka mēģinājuma laikos noslodze uz sistēmu nav absolūti vienmērīga, paralēli notiek sistēmas iekšējie procesi, kas var nedaudz ietekmēt rezultātu. Tāpēc tika pieprasījumi tika palaisti sistēmā vairākkārtīgi, lai testu sērija iezīmētu kopējo tendenci.



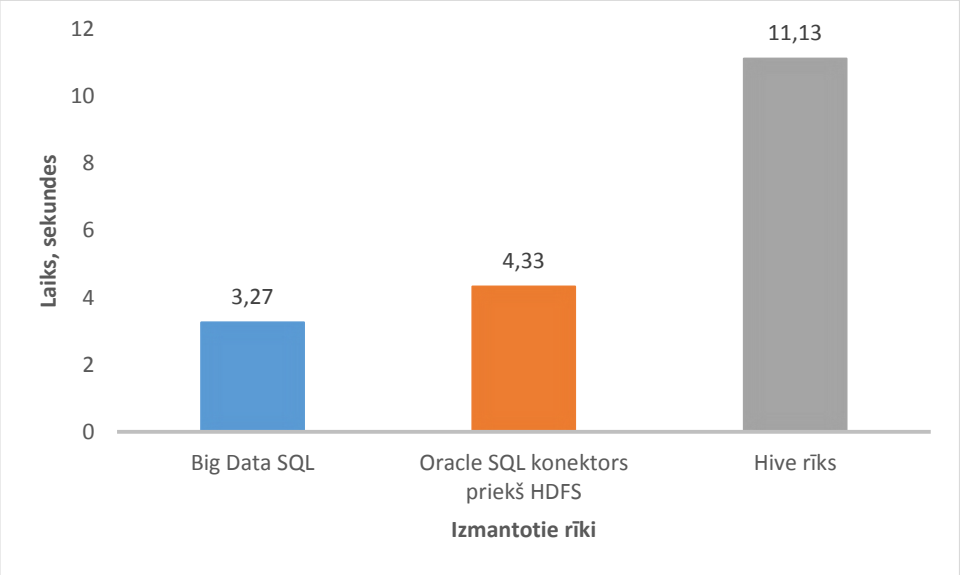
4.4.2..att. **Pirmā pieprasījuma testēšanas rezultātu vidējās vērtības**

Otrais pieprasījums ir līdzīgs pirmajam, tikai šajā gadījumā predikāts ar augstu selektivitāti ir *RDBMS* tabulā.



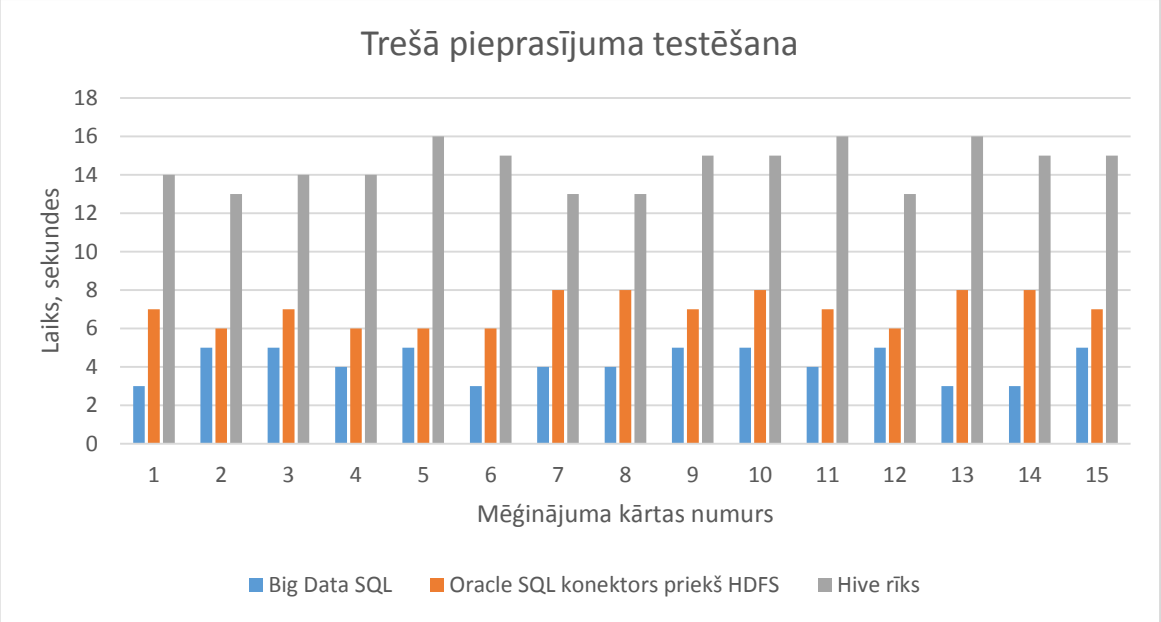
4.4.3.att. **Otrā pieprasījuma testēšanas rezultāti**

Attēlā 4.4.3. ir parādīti testēšanas rezultāti, ir redzams, ka tendence ir līdzīga, *Hive* rīks darbojas krietni lēnāk nekā *Oracle Big Data SQL* un *Oracle SQL* konektors. Taču tā rezultāts ir nedaudz labāks nekā pirmā testa rezultātos. Katra rīka vidējais laiks pieprasījuma veikšanai ir redzams attēlā 4.4.4. Arī pārējiem rīkiem laiks ir uzlabojies, to var izskaidrot ar to, ka *RDBMS* tabulas kolonna sr_item_sk, pēc kuras notika rezultātu atlasīšana ir indeksēta.



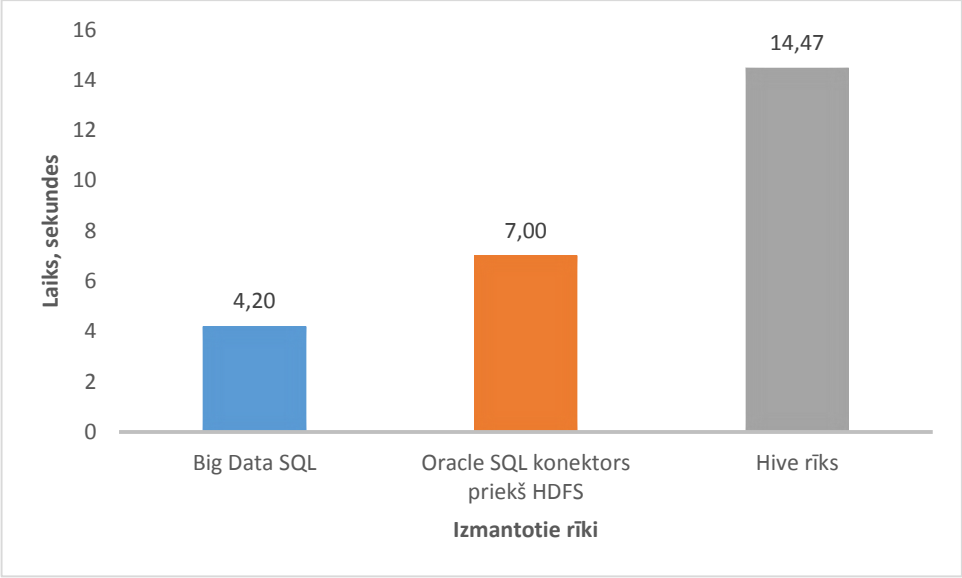
4.4.4.att. **Otrā pieprasījuma testēšanas rezultātu vidējās vērtības**

Trešajā pieprasījumā kā predikāts tika izmantota kolonna ar mazu selektivitāti ss_sales_price no STORE_SALES tabulas, attēlā 4.4.5. var redzēt iegūtos rezultātus pa mēģinājumiem. Kolonna ss_sales_price nav indeksēta un tā atrodas *HDFS* sistēmā. Rezultāti nedaudz atšķiras no iepriekšējā eksperimenta, bet ne būtiski. Ir redzams, ka arī šajā mēģinājumā vislabākais rezultāts ir *Oracle Big Data SQL*, tā vidējais ātrums veicot šāda veida datu savienošanu ir 4,20 sekundes. Tas ir par 8% lenāks nekā vidējais rezultāts veicot pirmo *SQL* pieprasījumu. Vidējie rīku rezultāti ir parādīti attēlā 4.4.6. Otrajā vietā ir *Oracle SQL* konektors priekš *HDFS*, kas arī datu apvienošanu veic *RDBMS* pusē. Šajā eksperimentā atšķirība starp *Oracle Big Data SQL* vidējo laiku un *Oracle SQL* konektora vidējo laiku ir jau 40%. Šo atšķirību var skaidrot ar to, ka šī pieprasījuma veikšanas laikā lielāks datu apjoms tiek nosūtīts no *HDFS* uz *RDBMS* sistēmu, rezultātā ir jāieraksta lielāks datu apjoms ārējās tabulās, kas patērē vairāk laika.



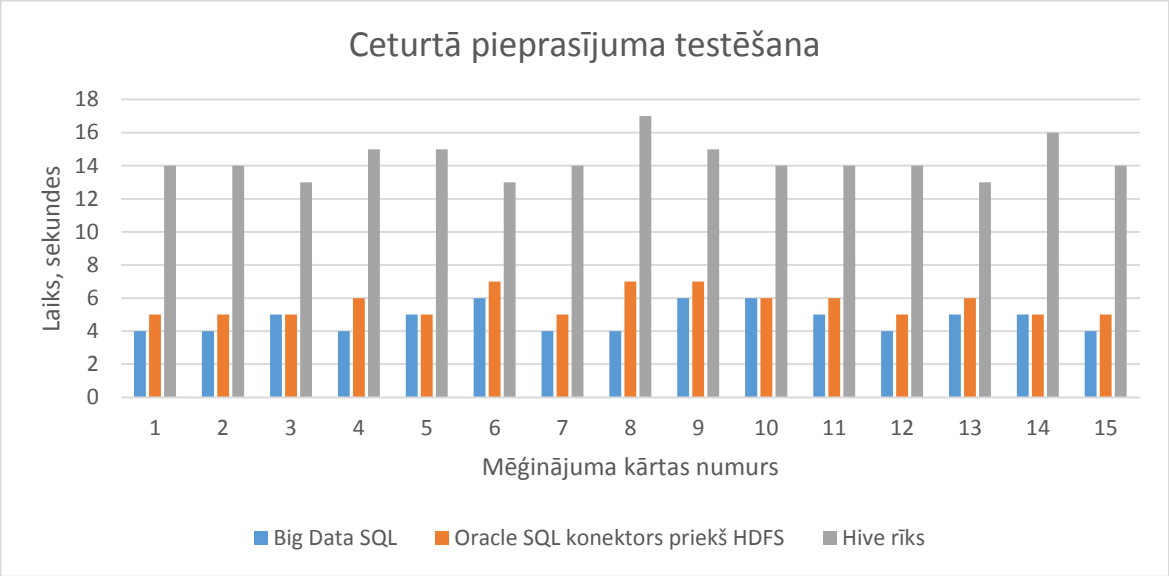
4.4.5.att. **Trešā pieprasījuma testēšanas rezultāti**

Arī šajā mēģinājumā *Hive* rīka rezultāts ir krietni lēnāks nekā pārējiem rīkiem. Kaut gan, ja skatās procentuāli, tad rezultātu atšķirība izmainījās tikai ar *Oracle Big Data SQL*, tad tā ir gandrīz 3,5 reizes. Savukārt, salīdzinot ar *Oracle* SQL konektoru priekš *HDFS*, var redzēt, ka procentuālā atšķirība ir palikusi apmēram tāda pati – *Hive* rīks ir apmēram 2 reizes lēnāks nekā *Oracle SQL* konektors. Lai arī savienojuma predikāts atrodas *HDFS* pusē, kolonna nav indeksēta, tāpēc ir jāizmanto pilna datu skenēšana, kas parasti aizņem vairāk laika nekā meklēšana pēc indeksa. Tā kā tabulā STORE_SALES ir gandrīz 3 miljardi ierakstu, tad arī pilna tabulas skenēšana aizņem daudz laika.



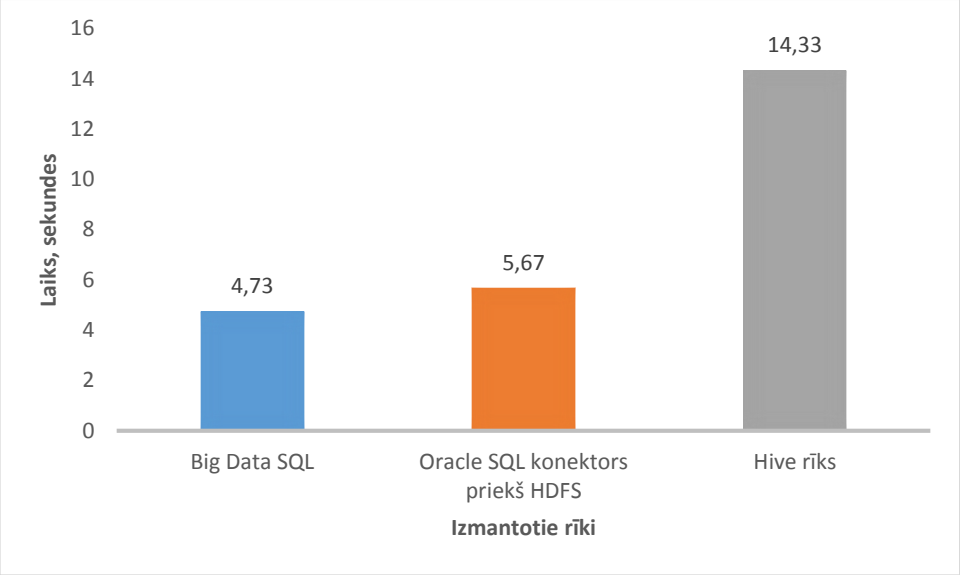
4.4.6.att. **Trešā pieprasījuma testēšanas rezultātu vidējās vērtības**

Ceturtajā pieprasījumā tiek meklēta informācija par pircējiem, kas atgrieza vairāk nekā vienu preci veikalā. Kā predikāts šajā pieprasījumā tiek izmantota sr_quantity kolonna, kas ir ar ļoti zemu selektivitāti, jo pamatā šīs kolonnas vērtība ir viens – pircēji lielākoties atgrieza pa vienai precei. Kolonna atrodas STORE_RETURNS tabulā *Oracle RDBMS* pusē un nav indeksēta, tāpēc arī šeit vajadzīgo vērtību meklēšanai ir nepieciešams veikt pilnu tabulas skenēšanu. Taču šajā tabulā atrodas daudz mazāk ierakstu nekā STORE_SALES tabulā, tuvu pie 288 miljoniem.



4.4.7.att. **Ceturtā pieprasījuma testēšanas rezultāti**

Testēšanas rezultāti pa mēģinājumiem ir attēloti 4.4.7. attēlā, var redzēt, ka atšķirība starp *Oracle Big Data SQL* rezultātu un *Oracle SQL* konektoru priekš *HDFS* ir samazinājusies, salīdzinot ar iepriekšējo eksperimentu. Tā kā lielākā daļa meklējamo datu atrodas *Oracle RDBMS* pusē, tad mazāk datu ir jāpārsūta no *HDFS* sistēmas, tādējādi samazinās ārējo tabulu izmantošana, līdz ar to arī samazinās laiks, kas nepieciešams datu savienojuma izveidošanai *Oracle SQL* konektoram. Vidējo datu savienojumu veidošanas laiku var aplūkot attēlā 4.4.8. Vidējā vērtība priekš *Oracle SQL* konektora šajā mēģinājumā ir 5,67, kas salīdzinot ar iepriekšējo testu ir par 19% mazāka. Taču joprojām lielāka nekā tā bija pirmā pieprasījuma testēšanas laikā, kad pieprasījumā kā predikāts tika lietota kolonna ar augstu selektivitāti.



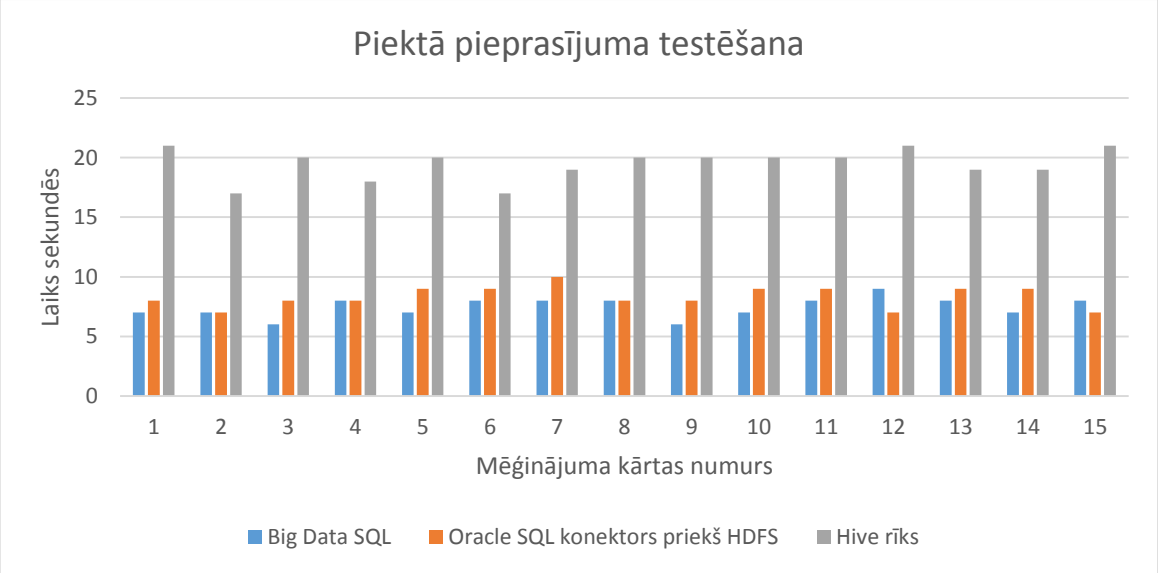
4.4.8.att. **Ceturtā pieprasījuma testēšanas rezultātu vidējās vērtības**

Rezultātu atšķirība starp *Big Data SQL* un *Oracle SQL* konektoru ir samazinājusies, ja salīdzina ar iepriekšējo testu. Šajā testā tā ir 17%. *Oracle Big Data SQL* vidējais laiks ir pieaudzis par 0,53 sekundēm, kas ir 12 %, salīdzinot ar iepriekšējā testa rezultātiem. Tas ir skaidrojams ar to, ka, lai arī datu apjoms, kas ir nepieciešams no *HDFS* sistēmas, ir samazinājies, testā kā predikāts tiek izmantota kolonna ar ļoti zemu selektivitāti, tāpēc arī skenēšanai nepieciešamais laiks ietekmēja rezultātu iegūšanas ātrumu.

Šajā testā *Hive* rīka rezultāts ir ļoti līdzīgs iepriekšējā pieprasījuma testēšanas rezultātam. Vidējā vērtība šajā testā ir 14,33 sekundes, kas ir par 1% labāks rezultāts nekā iepriekšējā testā. Tas norāda uz to, ka gadījumā, ja datu apvienošana notiek *HDFS* pusē, nav lielas atšķirības vai tabulas pilna skenēšana notiek *RDBMS* sistēmā vai *HDFS* sistēmā.

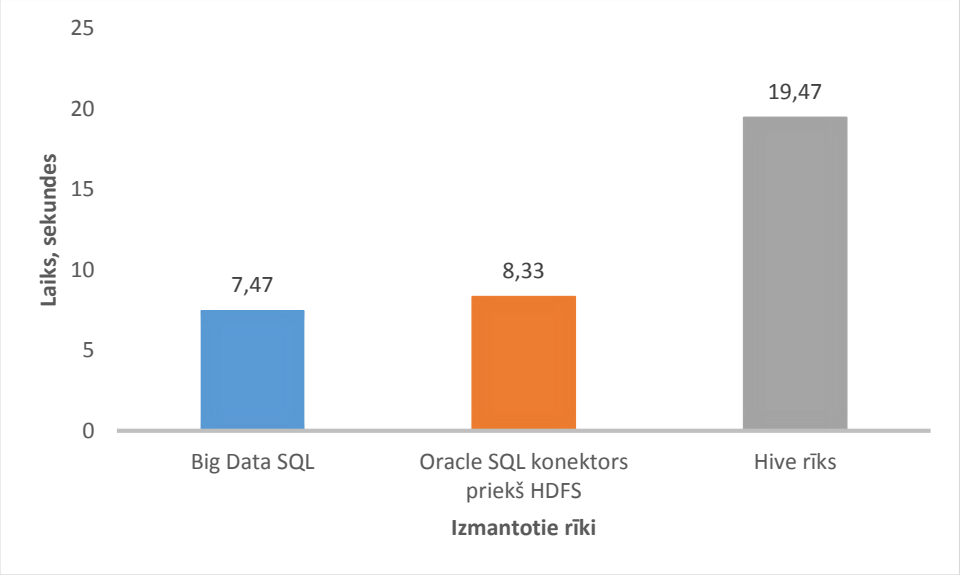
Piektajā pieprasījumā dati tiek apvienoti, kā arī tiek veikta agregācija, aprēķina, cik daudz preces tika atgrieztas katrā no veikaliem. Testa rezultāti pa mēģinājumiem ir apkopoti attēlā 4.4.9. Var redzēt, ka vairākos mēģinājumos *Oracle Big Data SQL* un *Oracle SQL* konektora rezultāti ir vienādi, kā piemēram, mēģinājumos 2, 4 un 8, kā arī dažos mēģinājumos *Oracle Big Data SQL* parāda sliktāku rezultātu, piemēra, mēģinājums 15 un 12. Taču, ja apskatās vidējās vērtības, kuras ir atrodamas 4.4.10.attēlā, tad ir redzams, ka *Oracle Big Data SQL* rezultāts ir par 10% labāks nekā *Oracle SQL* konektora rezultāts. Ja salīdzina šos rezultātus ar iepriekšējo testu rezultātiem, tad šī pieprasījuma veikšanai visiem rīkiem ir nepieciešams vairāk laika. Tas ir tāpēc, ka papildus datu apvienošanai ir nepieciešams veikt arī datu agregāciju. Jāņem vērā arī tas, ka šajā pieprasījumā nav

iesaistīti papildus predikāti, tāpēc nav nepieciešama sarežģīta datu atlase pēc papildus kritērijiem.



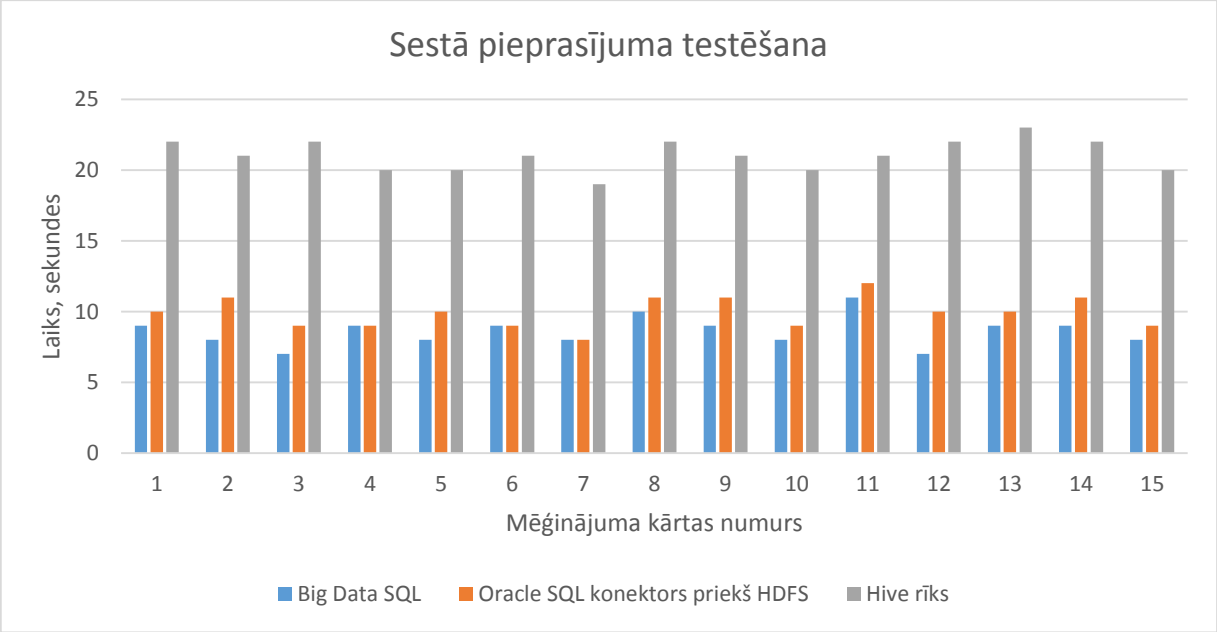
4.4.9.att. **Piektā pieprasījuma testēšanas rezultāti**

Ja salīdzina rezultātus ar ceturtā pieprasījuma testēšanas rezultātiem, tad *Big Data SQL* vidējais datu savienošanas laiks ir palielinājies par 37%, *Oracle SQL* konektora priekš *HDFS* laiks ir palielinājies uz 32% un *Hive* rīka laiks ir palielinājies par 26%. Savukārt atšķirība starp *Apache Hive* rīku un *Oracle Big Data SQL* ir kļuvusi mazāka, šī pieprasījuma testēšanas laikā tā ir tikai 2,6 reizes. Iepriekšējos testos *Oracle Big Data SQL* rīks ir bijis 3 līdz 3.5 reizes ātrāks. Arī atšķirība starp *Hive* un *Oracle SQL* konektoru ir samazinājusies, šī testa laikā *Apache Hive* ir bijis tikai 2,3 reizes lēnāks. Šie rezultāti parāda, ka vismazākā rīka vidējā darbības laika atkarība no pieprasījumu tipa ir *Hive* rīkam, lai arī tas darbojas lēnāk nekā citi rīki, tomēr tā darbība ir stabilāka neatkarīgi no uzdevuma, ko nepieciešams veikt.



4.4.10.att. **Piektā pieprasījuma testēšanas rezultātu vidējās vērtības**

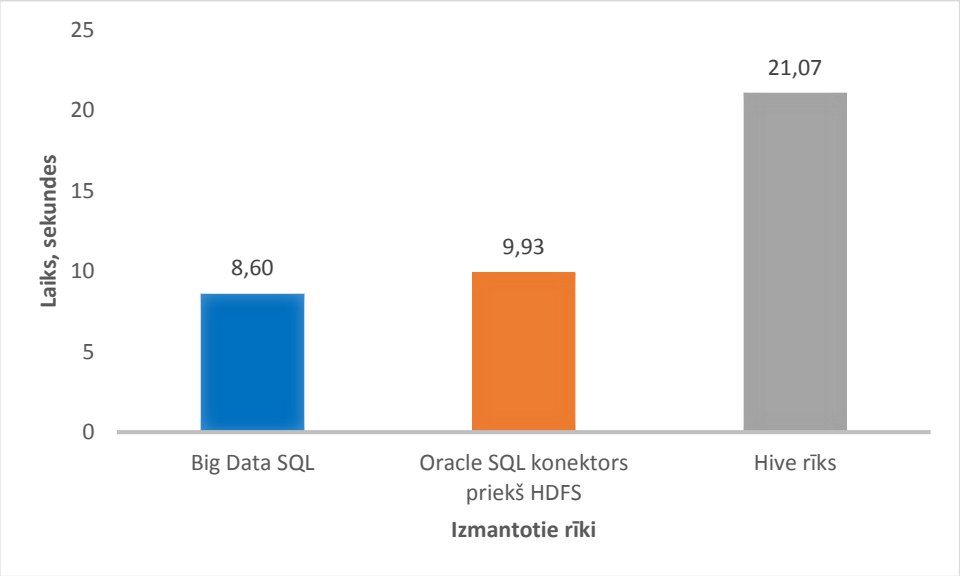
Sestajā pieprasījumā tika veikta datu apvienošana, izmantojot divus predikātus, kā arī veicot datu agregāciju. Abi predikāti ir ar zemu selektivitāti, viens ir sr_quantity kolonna STORE_RETURNS tabulā, kura atrodas *RDBMS* datubāzē, savukārt otrs predikāts ir HDFS sistēmas tabulā STORE_SALES uz ss_sales_price kolonnas. Savukārt agregācija tiek veikta izmantojot *RDBMS* STORE_RETURNS tabulas sr_store_sk kolonnu.



4.4.11.att. **Sestā pieprasījuma testēšanas rezultāti**

Pieprasījuma testēšanas rezultāti pa mēģinājumiem ir atainoti attēlā 4.4.11. Var redzēt, ka laiks, kas ir nepieciešams datu apvienošanai ir pieaudzis visiem rīkiem. Taču

atšķirība starp *Oracle Big Data SQL* un *Oracle SQL* konektoru priekš *HDFS* nav liela. Dažos mēģinājumos tie parāda vienādu rezultātu, kā piemēram, mēģinājumos 4, 6, 7. Testu vidējie rezultāti ir parādīti attēlā 4.4.12. Salīdzinot ar iepriekšējo pieprasījumu *Big Data SQL* datu apvienošanas un agregācijas ātrums ir samazinājies par 13%. Savukārt *Oracle SQL* konektora darbība ir palēninājusies par 11%. Tas ir palielinājis ātrdarbības atšķirību starp *Big Data SQL* un *Oracle SQL* konektoru no 10% līdz 13%, ja salīdzina ar iepriekšējā testa rezultātiem.

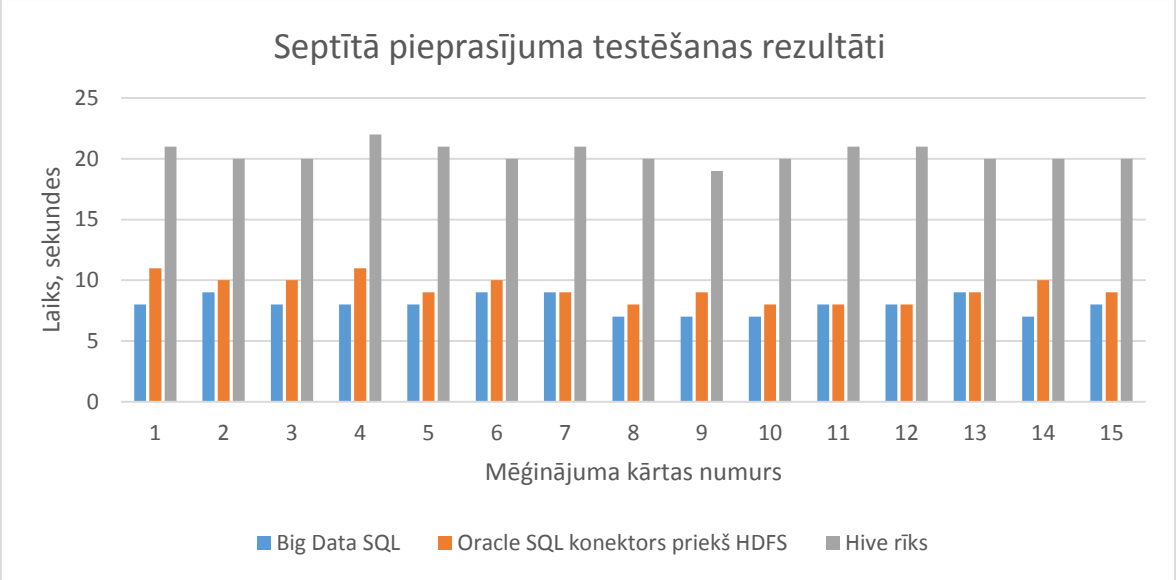


4.4.12.att. **Sestā pieprasījuma testēšanas rezultātu vidējās vērtības**

Hive rīka vidējais datu apvienošanas un agregācijas ātrums šī pieprasījuma testēšanas laikā bija 21,07, iepriekšējā pieprasījuma testēšanas laikā vidējā vērtība ir bijusi 19,47. Tas nozīmē, ka ātrums ir samazinājies par 8%. Arī šeit ir novērojams, ka *Hive* rīka darbības ātruma izmaiņas ir vismazākās no notestētajiem rīkiem.

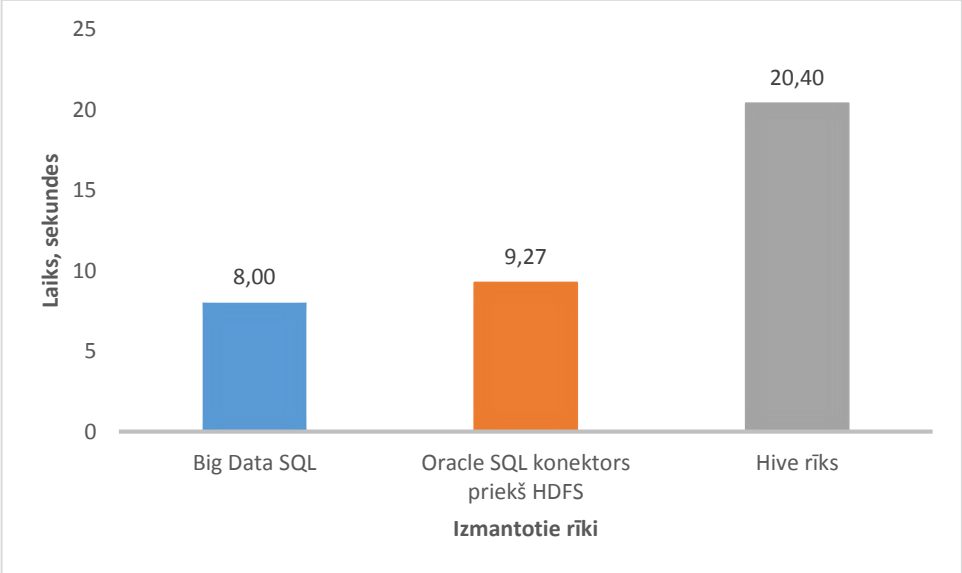
Sestā pieprasījuma testēšanas laikā visi rīki ir parādījuši ātrdarbības samazināšanos, kas ir izskaidrojams ar to, ka pieprasījumā tika izmantoti 2 predikāti ar zemu selektivitāti. Abas predikātos izmantotās kolonnas nav indeksētas, kas nozīmē, ka tika veikta pilna tabulas skenēšana, lai atrastu vajadzīgās vērtības.

Septītais pieprasījums testē datu apvienošanas gadījumu, kad tiek lietoti divi predikāti ar augstu selektivitāti, viens predikāts uz *RDBMS* tabulas, savukārt otrs uz *HDFS* tabulas, kā arī tiek piemērota rezultātu agregācija. Šī testa rezultāti ir redzami attēlā 4.4.13. Pieprasījuma struktūra ir līdzīga iepriekšējā pieprasījuma struktūrai, atšķiras tikai predikātu selektivitāte.



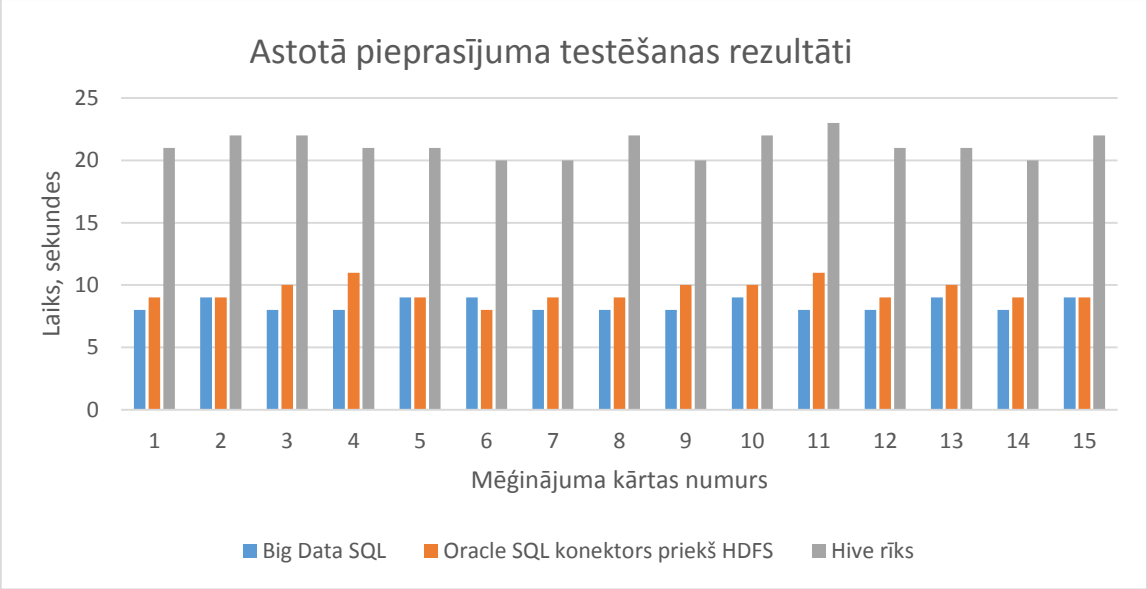
4.4.13.att. **Septītā pieprasījuma testēšanas rezultāti**

Arī rezultāti ir līdzīgi iepriekšējā testa rezultātiem, bet visu rīku pieprasījumu izpildes laiks ir uzlabojies. *Big Data SQL* un *Oracle SQL* konektora priekš *HDFS* vidējais izpildes laiks ir uzlabojies par 7%, savukārt *Apache Hive* rīka laiks ir uzlabojies par 4%. Vidējais laiks, ko patērēja katrs no rīkiem testējot septīto pieprasījumu ir parādīts attēlā 4.4.14.



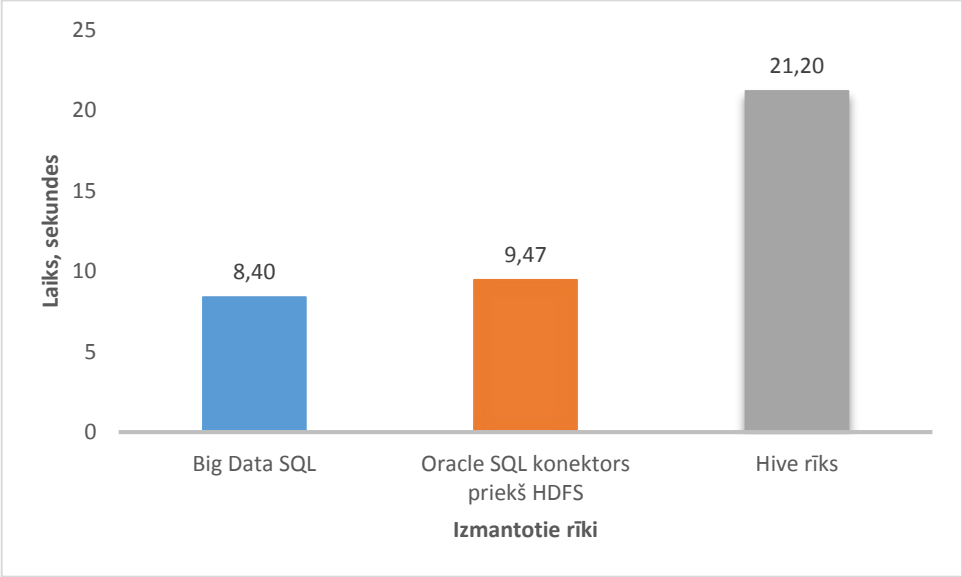
4.4.14.att. **Septītā pieprasījuma testēšanas rezultātu vidējās vērtības**

Astotajā pieprasījumā tika testēts gadījums, kad predikāts *HDFS* tabulā ir ar augstu selektivitāti, bet predikāts, ko piemēro *RDBMS* tabulai ir ar zemu selektivitāti, kā arī tiek pildīta agregācija. Testa rezultāti pa mēģinājumiem ir apkopoti attēlā 4.4.15.



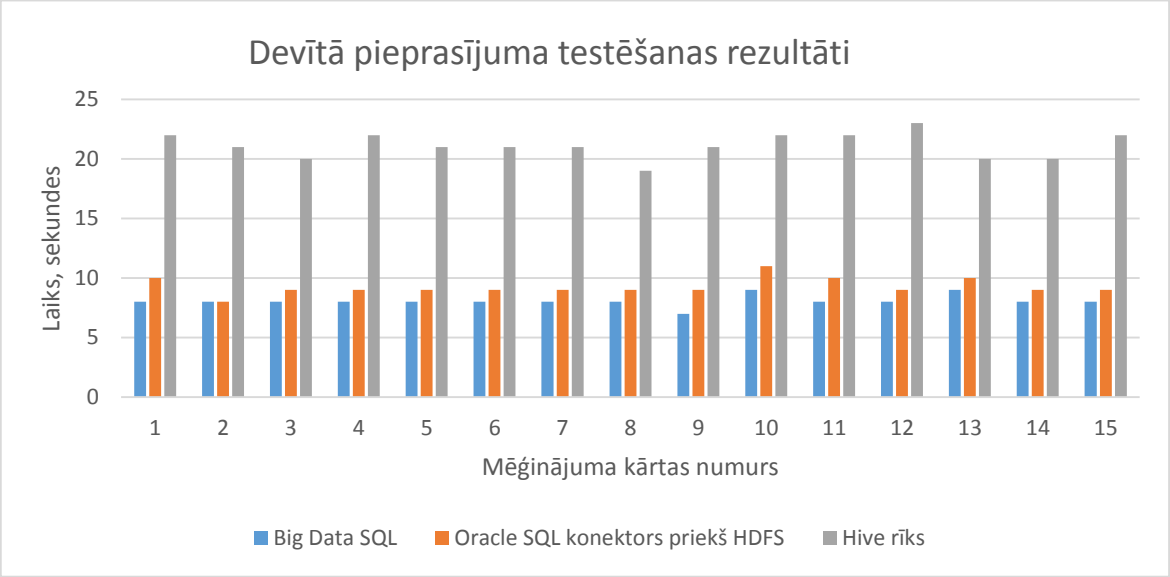
4.4.15.att. **Astotā pieprasījuma testēšanas rezultāti**

No rezultātiem ir redzams, ka izpildes laiks katram rīkam ir palielinājies. *Apache Hive* rīkam tas ir sasniedzis vērtību 21,20 sekundes, šis rezultāts ir pat par 1% lielāks nekā gadījumā, kad abi predikāti ir bijuši ar zemu selektivitāti. Taču šī starpība nav ļoti liela un, iespējams ka, uz rezultātu atstāja ietekmi kopējā sistēmas noslodze testa laikā. Vidējās laika vērtības katram no rīkiem ir parādītas attēlā 4.4.16. *Oracle* rīku vidējie laiki arī pieauga salīdzinot ar septītā testa rezultātiem, taču tie ir mazāki nekā sestā pieprasījuma testēšanas laikā.



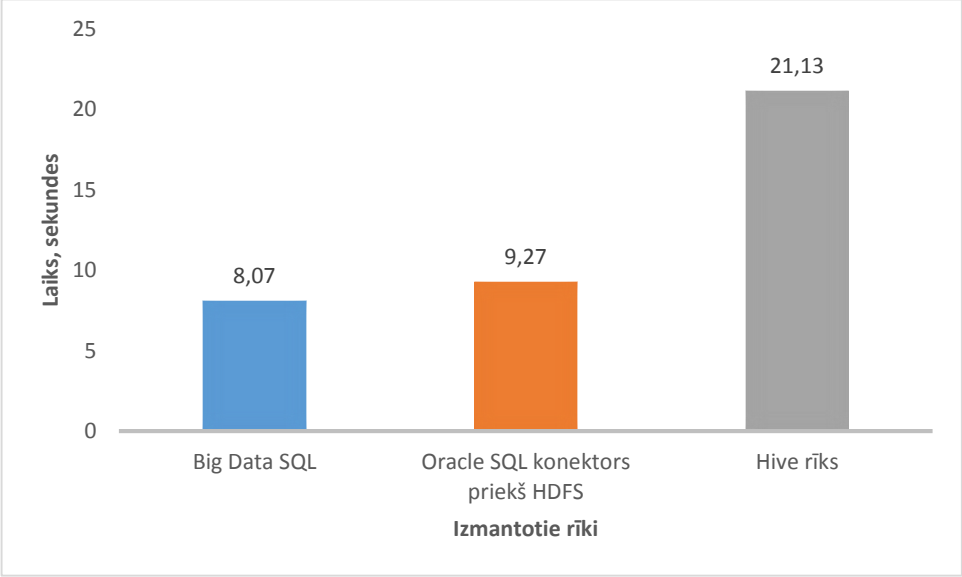
4.4.16.att. **Astotā pieprasījuma testēšanas rezultātu vidējās vērtības**

Devītais pieprasījums ir līdzīgs astotajam, bet šajā gadījumā predikāts ar zemu selektivitāti ir uz *HDFS* tabulas, bet predikāts ar augstu selektivitāti ir uz *RDBMS* tabulas. Rezultātu apkopojums pa mēģinājumiem ir parādīts attēlā 4.4.17.



4.4.17.att. **Devītā pieprasījuma testēšanas rezultāti**

Šajā gadījumā *RDBMS* tabulas predikāts tiek piemērots kolonnai, kas ir indeksēta, tāpēc arī rezultāts visiem rīkiem ir labāks nekād iepriekšējā testā. *Oracle Big Data SQL* rezultāts ir uzlabojies par 4%, savukārt *Oracle SQL* konektors priekš *HDFS* rezultāts ir kļuvis labāks par 2%.



4.4.18.att. **Devītā pieprasījuma testēšanas rezultātu vidējās vērtības**

Testu rezultātā tika noskaidrots, ka vislabākos rezultātus ātrdarbībā visi rīki parāda, ja predikāti ir ar augstu selektivitāti. Pretējā gadījumā ir novērota visu rīku veiktspējas

degradācija. Visslikākie rezultāti tika iegūti veicot pieprasījums ar zemas selektivitātes predikātiem un agregāciju. Tā kā *Oracle Big Data SQL* izmanto *SmartScan* funkcionalitāti un Blūma filtrus, tad testu rezultāti šim rīkam ir vislabākie, jo tajā ir iebūvētas papildus tehnoloģijas, kas ļauj samazināt pārsūtītu datu apjomu, kā arī atlasīt *HDFS* neindeksētos datus efektīvāk, apejot pilnu tabulas skenēšanas nepieciešamību. Gadījumos, kad *HDFS* tabulas ir ļoti lielas šī funkcionalitāte ļauj *Oracle Big Data SQL* veikt datu apvienošanu ievērojami ātrāk nekā citi rīki.

Datu apvienošanas veiktspēja hibrīdās sistēmās ir apskatīta arī citos pētījumos. 2015.gada *IBM* pētījumā par hibrīdo datu noliktavu veiktspēju arī tika pētītas datu apvienošanas iespējas. Tika testēti gadījumi, kad datu apvienošana notiek *RDBMS* pusē (kā relāciju datubāze šajā pētījumā tika izmantota *IBM DB2*) un, kad datu apvienošana notiek *HDFS* pusē, kā arī Blūma filtru pielietošanas ietekme uz rezultātu. Pētījumā iegūtie rezultāti ir līdzīgi šajā darbā iegūtajiem rezultātiem, protams, izpildes laiki ir atšķirīgi, jo testi tika izpildīti uz atšķirīgām datu kopā, kā arī izmantotie *SQL* pieprasījumi nav identiski, taču rezultātā iegūtās tendences ir līdzīgas. Pētījuma laikā tika atklāta Blūma filtra izmantošanas pozitīvā ietekme uz pieprasījumu veiktspēju, īpaši, ja predikātu selektivitāte ir augsta. Arī šajā darbā ir redzama šāda sakarība, *Oracle Big Data SQL*, kas savā darbībā izmanto Blūma filtrus, ir parādījis vislabākos rezultātus visos pieprasījumu testu gadījumos, vislabākie rezultāti tika iegūti, kad tika testēti pieprasījumi ar augstu selektivitāti predikātos.

Tika izpētīta arī cita tendence – datu apvienošanas veiktspēja *RDBMS* pusē bez Blūma filtriem, pie augstas predikātu selektivitātes ir līdzīga datu apvienošanai ar Blūma filtriem. Taču, samazinoties predikātu selektivitātei, atšķirība starp datu apvienošanas veiktspēju *RDBMS* pusē ar Blūma filtru un bez Blūma filtra palielinās. To var novērot arī šajā darbā veiktajos eksperimentos. Piemēram, atšķirība starp *Oracle Big Data SQL* veikstpēju un *Oracle SQL* konektora priekš *HDFS* veiktspēju pie augstas selektivitātes pirmajā pieprasījumā bija 25%, bet trešajā pieprasījumā, kur tika izmantots predikāts ar zemu selektivitāti, veiktspējas atšķirība bija 40%. *IBM* pētījumā datu savienojumi *RDBMS* pusē tika testēti izmantojot *IBM Big SQL* rīku, kas arī ļauj ar *SQL* pieprasījumu izvadīt datus no *HDFS* sistēmas. *IBM Big SQL* neizmanto Blūma filtrus datu atlasē un apvienošanā, pētījumā tika izveidota papildus programma, kas piemēroja *IBM Big SQL* rezultātiem Blūma filtrus. Tāpēc *IBM Big SQL* rīka darbības algoritms datu apvienošanā ir līdzīgs *Oracle SQL* konektora priekš *HDFS* darbībai.

IBM pētījumā tika atklāts, ka broadcast datu apvienošanai *HDFS* pusē ir limitēta lietojamība, jo šajā datu apvienošanas veidā diezgan liels datu apjoms tiek pārsūtīts starp mezgliem, kas nav efektīvi. Arī šī darba ietvaros tika konstatēts, ka *Apache Hive* rīka veiktspēja ir zemāka nekā *Oracle* rīkiem.

Veiktsspēja nav vienīgais kritērijs, pēc kura tiek izvēlēts kādu produktu lietot datu noliktavā. Lai arī testu laikā vislabāko veiktspējas rezultātu ir uzrādījis *Oracle Big Data SQL* produkts, tā lietošanai ir ierobežojumi, kā piemēram, operētājsistēmu skaits, ar kuru *Oracle Big Data SQL* ir sertificēts ir ļoti ierobežots, kā arī nepieciešami daudzi papildus produkti, lai *Oracle Big Data SQL* strādātu. Tāpēc šī rīka ieviešana prasa lielus ieguldījumus infrastruktūrā. Vērtējot pēc šāda kritērija, *Oracle SQL* konektoru priekš *HDFS* ir iespējams ieviest jau esošajā sistēmā, jo tā ieviešanai ir daudz mazāk infrastruktūras prasību.

SECINĀJUMI

Maģistra darba ietvaros tika izpētīti *Oracle* un citu lielāko ražotāju piedāvātie risinājumi *Big Data* apstrādei, pamatā visu piegādātāju risinājumus var sadalīt integrācijas risinājumos, Big data analīzes un vizualizācijas risinājumos un risinājumos, kas ir balstīti uz mākoņpakalpojumiem. Tā kā *Hadoop* un *MapReduce* sevi ir pierādījuši kā efektīvu risinājumu *Big Data* apstrādē, citi ražotāji savus produktus balsta uz šīm tehnoloģijām.

Padziļināti tika pētīta datu apvienošanas iespēja hibrīdajā datu noliktavā, kur daļa no datiem tiek glabāti *HDFS* sistēmā un daļa *RDBMS* sistēmā. Eksperimentu laikā tika secināts, ka:

- Pie predikātu augstas selektivitātes, neatkarīgi no tā vai predikāts atrodas *HDFS* vai *RDBMS* sistēmā, datu apvienošana ar *Oracle Big Data SQL* un Oracle *SQL* priekš *HDFS* rīkiem uzrāda līdzīgus rezultātus.;
- Pie predikātu zemas selektivitātes, *Oracle Big data SQL* veikstpēja kļūst ievērojami labāka, nekā *Oracle SQL* konektora priekš *HDFS* veikspēja, jo *Oracle Big Data SQL* savā darbā izmanto Blūma filtru, kas palīdz samazināt pārsūtīto datu apjomu;
- Lai uzlabotu *Oracle SQL* konektora priekš *HDFS* veikstpēju, gadījumā, ja pieprasījumos tiek izmantoti predikāti ar zemu selektivitāti, atbilstošās kolonnas ir jāindeksē. Vai arī jāveido savs risinājums, kas papildina *Oracle SQL* konektora priekš *HDFS* darbību un piemēro Blūma filtrus datu atlases laikā;
- *Apache Hive* rīka darbības veiktspēja hibrīdās datu noliktavas gadījumā ir krietni lēnāka, nekā pārējo testēto rīku veiktspēja, taču šī rīka veikspējas rezultāti ir stabili – maz mainās atkarībā no predikātu selektivitātes un agregācijas izmantošanas;
- Lai uzlabotu *Apache Hive* rīka veiktspēju, ir nepieciešams veidot savu risinājumu, kas papildinātu šī rīka darbību ar Blūma filtru lietošanu;
- *Broadcast* tipa datu apvienošanai ir būtisks trūkums – liels pārsūtīto datu apjoms starp sistēmām. Tāpēc šāda veida datu apvienošanai ir ierobežota lietojamība un šis risinājums neder gadījumos, ja ir nepieciešams apvienot lielas datu kopas;

- Būtisks trūkums *HDFS* sistēmā glabātajiem datiem – tie nav indeksēti, tāpēc daudz laika pieprasījumu izpildē tiek patērēts uz tabulu pilnīgu skenēšanu, kas arī ietekmē rīku darbības veiktspēju. *Oracle Big Data SQL* tehnoloģijā šī problēma tiek risināta ar *SmartScan* funkcionalitāti, kas darbojas kā diska indekss, līdz ar to nepieciešamie dati tiek atrasti ātrāk;
- Agregācijas izmantošana pieprasījumā palēnina pieprasījumu izpildes laiku, tā kā datu noliktavās lielu datu kopu agregācijas ir biežas, tad rīka ātrdarbība ir viens no būtiskajiem faktoriem.

Paveikto testu rezultāti nav pilnīgi, lai viennozīmīgi izdarītu secinājumus par datu apvienošanas veidu veiktspēju, jo netika testēta rīku veiktspēja pie dažādiem datu formātiem *HDFS* sistēmā. Šāda veida papildus pētījumi varētu sniegt papildus informāciju par datu apvienošanas iespējām hibrīdās sistēmās, kā arī par testēto rīku iespējām.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] B. Marr, *Big Data Using SMART Big Data, Analytics and Metrics To Make Better Decisions and Improve Performance*, Wiley, 2015, p.256

[2] K.Krishnan, *Data Warehousing in the Age of Big Data*, Elsevier Science, Morgan Kaufmann, 2013, p.344

[3] S. Mohanty, *Big Data Imperatives Enterprise Big DataWarehouse, BI Implementations and Analytics*, Apress, 2013, p.320

[4] F.Chen, M.Hsu, "A Performance Comparison of Parallel DBMSs and Map Reduce on Large-Scale Text Analytics", HP Labs, Mar. 2013

[5] K.Bajda-Pawlikowski, D.J.Abadi, A.Silberschatz, E.Paulson, "Efficient Processing of Data Warehouse Queries in a Split Execution Environment", Hadapt Inc.,Yale University, University of Wisconsin-Madison, 2011

[6] M.Muddasir, R.H C, "Comparing Implementation Features on Map Reduce in RDBMS with Distributed Cluster", International Journal of Computer Applications, ICCTAC 2015(2), May 2015, pp. 19-24

[7] Peter C., D.Manolios, P.Manolios, "Bloom Filters in Probabilistic Verification",Georgia Institute of Technology, 2004

[8] X.Su, G.Swart, "Oracle In-Database Hadoop: When MapReduce Meets RDBMS", Oracle Corporation, 2012

[9] Y.Tian, T.Zou, F.Ozcan, R.Goncalves, H.Piranesh, "Joins for Hybrid Warehouses: Exploiting Massive Parallelism in Hadoop and Enterprise Data Warehouses", IBM Almaden Research Center USA, Google Inc, 2015

[10] Dean, J., & Ghemawat, "MapReduce: a data processing tool", Communications of the ACM, Jan. 2010,pp. 72–77.

[11] "2016 Gartner Magic Quadrant for BI and Analytics findings" Research, Gartner Inc., Feb. 2016

[12] "Analytics Platform System Appliance Update 4 Documentation and Client", Microsoft, Dec. 2015

[13] "Apache Hive Getting Started Guide", Apache Hive TM, Feb. 2016

[14] "Cloudera Distribution for Hadoop for Teradata Administrator Guide Release 5.4, 5.5", Teradata Corp. Dec. 2015

[15] "Deployment Step-by-step instructions to deploy Oracle Big Data Lite Virtual Machine Version 4.4.0", Oracle Inc, Feb. 2016

[16] "HDFS Architecture Guide", Apache Hadoop, Apr. 2013

[17] "Hive SQL Language Manual", Apache Hive TM, Oct. 2015

[18] "Hortonworks Data Platform for Teradata Administrators Guide Release 2.3, 2.4", Teradata Corp. Mar. 2016

[19] "IBM BigInsights 4.1 documentation" IBM corporation, Dec. 2015

[20] "Installation Guide for Ambari 2.2.2", Apache Ambari, Apr. 2016

[21] "Microsoft HDInsight documentation", Microsoft, Mar. 2016

[22] "Oracle Big Data Connectors User's Guide Release 4 (4.4)", Oracle Corp., Jan. 2016

[23] "Oracle Big Data Discovery Data Processing Guide, Version 1.1.1", Oracle Corp., Oct. 2015

[24] "Oracle Big Data Discovery User Guide", Oracle Corp., Dec. 2015

[25] "Oracle Big Data SQL User's Guide Release 3 (3.0)", Oracle Corp., Jan. 2016

[26] "Oracle Fusion Middleware Administering Oracle GoldenGate for Big Data, Release 12c (12.2.0.1)", Oracle Corp., Dec. 2015

[27] "TPC BENCHMARK DS Standard Specification Version 2.1.0", Transaction Processing Performance Council, Nov. 2015

[28] "Big Data: Trends, Strategies, and SAP Technology" White Paper, SAP corp., Jul. 2012

[29] "Data Warehouse Optimization with Hadoop" White Paper, Cloudera, Jan. 2014

[30] "Hadoop and the Data Warehouse:When to Use Which" White Paper, Teradata Corp, Cloudera Inc., Mar. 2016

[31] "In-Database Map-Reduce" White Paper, Oracle Inc, Oct.2009

[32] "Microsoft SQL Server, Optimizing Distributed Database Design for Analytics Platform System" White Paper, Microsoft, Jun. 2015

[33] "Oracle: Big Data for Enterprise" White Paper, Oracle Corp., Sep. 2014

[34] "Oracle Real Application Clusters (RAC)" White Paper,, Oracle Inc, Jun. 2013

[35] "SAP HANA System Landscape Guide" White Paper, SAP corp., Jan. 2014

[36] "Technical Overview SAP HANA" White Paper,, SAP corp., Mar. 2014

[37] "Unified Query for Big Data Management Systems,Integrating Big Data Systems with Enterprise Data Warehouses" White Paper, Oracle Corp., Jan. 2015

PIELIKUMI

1.pielikums. STORE_SALES tabulas definīcija

Kolonna	Datu tips	NULLS	Primārā atslēga	Sekundārā atslēga
ss_sold_date_sk	identifier	-	-	d_date_sk
ss_sold_time_sk	identifier	-	-	t_time_sk
ss_item_sk (1)	identifier	N	Y	i_item_sk
ss_customer_sk	identifier	-	-	c_customer_sk
ss_cdemo_sk	identifier	-	-	cd_demo_sk
ss_hdemo_sk	identifier	-	-	hd_demo_sk
ss_addr_sk	identifier	-	-	ca_address_sk
ss_store_sk	identifier	-	-	s_store_sk
ss_promo_sk	identifier	-	-	p_promo_sk
ss_ticket_number (2)	identifier	N	Y	-
ss_quantity	integer	-	-	-
ss_wholesale_cost	decimal(7,2)	-	-	-
ss_list_price	decimal(7,2)	-	-	-
ss_sales_price	decimal(7,2)	-	-	-
ss_ext_discount_am	decimal(7,2)	-	-	-
ss_ext_sales_price	decimal(7,2)	-	-	-
ss_ext_wholesale_cost	decimal(7,2)	-	-	-
ss_ext_list_price	decimal(7,2)	-	-	-
ss_ext_tax	decimal(7,2)	-	-	-
ss_coupon_amt	decimal(7,2)	-	-	-
ss_net_paid	decimal(7,2)	-	-	-
ss_net_paid_inc_tax	decimal(7,2)	-	-	-
ss_net_profit	decimal(7,2)	-	-	-

2.pielikums. STORE_RETURNS tabulas definīcija

Kolonna	Datu tips	NULLS	Primārā atslēga	Sekundārā atslēga
sr_returned_date_sk	identifier	-	-	d_date_sk
sr_return_time_sk	identifier	-	-	t_time_sk
sr_item_sk (1)	identifier	N	Y	i_item_sk,ss_item_sk
sr_customer_sK	identifier	-	-	c_customer_sk
sr_cdemo_sk	identifier	-	-	cd_demo_sk
sr_hdemo_sk	identifier	-	-	hd_demo_s
sr_addr_sk	identifier	-	-	ca_address_sk
sr_store_sk	identifier	-	-	s_store_sk
sr_reason_sk	identifier	-	-	r_reason_sk
sr_ticket_number (2)	identifier	N	Y	ss_ticket_number
sr_return_quantity	integer	-	-	-
sr_return_amt	decimal(7,2)	-	-	-
sr_return_tax	decimal(7,2)	-	-	-
sr_return_amt_inc_tax	decimal(7,2)	-	-	-
sr_fee	decimal(7,2)	-	-	-
sr_return_ship_cost	decimal(7,2)	-	-	-
sr_refunded_cash	decimal(7,2)	-	-	-
sr_reversed_charge	decimal(7,2)	-	-	-
sr_store_credit	decimal(7,2)	-	-	-
sr_net_loss	decimal(7,2)	-	-	-

3.pielikums. Pieprasījumu testēšanās gaitā iegūtie rezultāti

1. Pirmā pieprasījuma testēšanas rezultāti

Mēģinājuma Nr.	Laiks datu apvienošanai, sekundēs		
	Big Data SQL	Oracle SQL konektors priekš HDFS	Hive rīks
1	3	5	9
2	5	4	10
3	4	5	13
4	3	6	11
5	3	6	10
6	5	6	13
7	7	5	13
8	2	6	12
9	4	5	13
10	3	4	10
11	4	4	11
12	4	5	9
13	3	5	13
14	5	6	13
15	3	6	12

2. Otrā pieprasījuma testēšanas rezultāti

Mēģinājuma Nr.	Laiks datu apvienošanai, sekundēs		
	Big Data SQL	Oracle SQL konektors priekš HDFS	Hive rīks
1	3	4	11
2	4	4	10
3	3	5	10
4	3	5	11

5	3	5	11
6	3	4	12
7	4	4	11
8	3	4	11
9	3	4	10
10	3	4	13
11	4	5	11
12	3	4	11
13	3	4	12
14	4	4	11
15	3	5	12

3. Trešā pieprasījuma testēšanas rezultāti

Mēģinājuma Nr.	Laiks datu apvienošanai, sekundēs		
	Big Data SQL	Oracle SQL konektors priekš HDFS	Hive rīks
1	3	7	14
2	5	6	13
3	5	7	14
4	4	6	14
5	5	6	16
6	3	6	15
7	4	8	13
8	4	8	13
9	5	7	15
10	5	8	15
11	4	7	16
12	5	6	13
13	3	8	16
14	3	8	15
15	5	7	15

4. Ceturtā pieprasījuma testēšanas rezultāti

Mēģinājuma Nr.	Laiks datu apvienošanai, sekundēs		
	Big Data SQL	Oracle SQL konektors priekš HDFS	Hive rīks
1	4	5	14
2	4	5	14
3	5	5	13
4	4	6	15
5	5	5	15
6	6	7	13
7	4	5	14
8	4	7	17
9	6	7	15
10	6	6	14
11	5	6	14
12	4	5	14
13	5	6	13
14	5	5	16
15	4	5	14

5. Piektā pieprasījuma testēšanas rezultāti

Mēģinājuma Nr.	Laiks datu apvienošanai, sekundēs		
	Big Data SQL	Oracle SQL konektors priekš HDFS	Hive rīks
1	7	8	21
2	7	7	17
3	6	8	20
4	8	8	18

5	7	9	20
6	8	9	17
7	8	10	19
8	8	8	20
9	6	8	20
10	7	9	20
11	8	9	20
12	9	7	21
13	8	9	19
14	7	9	19
15	8	7	21

6. Sestā pieprasījuma testēšanas rezultāti

Mēģinājuma Nr.	Laiks datu apvienošanai, sekundēs		
	Big Data SQL	Oracle SQL konektors priekš HDFS	Hive rīks
1	9	10	22
2	8	11	21
3	7	9	22
4	9	9	20
5	8	10	20
6	9	9	21
7	8	8	19
8	10	11	22
9	9	11	21
10	8	9	20
11	11	12	21
12	7	10	22
13	9	10	23
14	9	11	22
15	8	9	20

7. Septītā pieprasījuma testēšanas rezultāti

Mēģinājuma Nr.	Laiks datu apvienošanai, sekundēs		
	Big Data SQL	Oracle SQL konektors priekš HDFS	Hive rīks
1	8	11	21
2	9	10	20
3	8	10	20
4	8	11	22
5	8	9	21
6	9	10	20
7	9	9	21
8	7	8	20
9	7	9	19
10	7	8	20
11	8	8	21
12	8	8	21
13	9	9	20
14	7	10	20
15	8	9	20

8. Astotā pieprasījuma testēšanas rezultāti

Mēģinājuma Nr.	Laiks datu apvienošanai, sekundēs		
	Big Data SQL	Oracle SQL konektors priekš HDFS	Hive rīks
1	8	9	21
2	9	9	22
3	8	10	22
4	8	11	21
5	9	9	21
6	9	8	20

7	8	9	20
8	8	9	22
9	8	10	20
10	9	10	22
11	8	11	23
12	8	9	21
13	9	10	21
14	8	9	20
15	9	9	22

9. Devītā pieprasījuma testēšanas rezultāti

Mēģinājuma Nr.	Laiks datu apvienošanai, sekundēs		
	Big Data SQL	Oracle SQL konektors priekš HDFS	Hive rīks
1	8	10	22
2	8	8	21
3	8	9	20
4	8	9	22
5	8	9	21
6	8	9	21
7	8	9	21
8	8	9	19
9	7	9	21
10	9	11	22
11	8	10	22
12	8	9	23
13	9	10	20
14	8	9	20
15	8	9	22

4.pielikums. Oracle SQL konektora priekš HDFS konfigurācijas fails ārējās tabulas izveidei

```
<?xml version="1.0"?>
<!-- Required Properties -->
<configuration>
  <property>
    <name>oracle.hadoop.exttab.tableName</name>
    <value>S.STORE_SALES</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.defaultDirectory</name>
    <value>STORE_SALES_DIR </value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.dataPaths</name>
    <value>/data/s1/*.csv,/data/s2/*.csv</value>
  </property>

<!-- Use either columnCount or columnNames -->

  <property>
    <name>oracle.hadoop.exttab.columnCount</name>
    <value>23</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.url</name>
    <value>jdbc:oracle:thin:@//myhost:1521/RAC</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.user</name>
    <value>hadoop</value>
  </property>
```

5.pielikums. Oracle Big Data SQL ārējās tabulas veidošanas pilns komandas teksts

```
CREATE TABLE store_sales        (ss_sold_date_sk VARCHAR2(10),
                                 ss_sold_time_sk VARCHAR2(10),
                                 ss_item_sk  VARCHAR2(10),
                                 ss_customer_sk VARCHAR2(10),
                                 ss_cdemo_sk VARCHAR2(10),
                                 ss_hdemo_sk VARCHAR2(10),
                                 ss_addr_sk VARCHAR2(10),
                                 ss_store_sk VARCHAR2(10),
                                 ss_promo_sk VARCHAR2(10),
                                 ss_ticket_number VARCHAR2(10),
                                 ss_quantity NUMBER,
                                 ss_wholesale_cost NUMBER(7,2),
                                 ss_list_price NUMBER(7,2),
                                 ss_sales_price NUMBER(7,2),
                                 ss_ext_discount_am NUMBER(7,2),
                                 ss_ext_sales_price NUMBER(7,2),
                                 ss_ext_wholesale_cost NUMBER(7,2),
                                 ss_ext_list_price NUMBER(7,2),
                                 ss_ext_tax NUMBER(7,2),
                                 ss_coupon_amt NUMBER(7,2),
                                 ss_net_paid NUMBER(7,2),
                                 ss_net_paid_inc_tax NUMBER(7,2),
                                 ss_net_profit NUMBER(7,2) )

   ORGANIZATION EXTERNAL
   (
     TYPE oracle_hdfs
     DEFAULT DEFAULT_DIR

  LOCATION ('hdfs:/data/s1/*.csv'));
```

DOKUMENTĀRĀ LAPA

Maģistra darbs: **ORACLE RISINĀJUMU PIELIETOJUMS BIG DATA KONTEKSTĀ**

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: _____
(Autora paraksts)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par **p i e m ē r o t u / n e p i e m ē r o t u** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Darba vadītājs: _____
(Vadītāja paraksts)

Darbs iesniegts **maģistrantūras sekretariātā** _____.
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: _____.
(Metodiķes paraksts)

Recenzents: _____
(Akad.amats, zin.grāds, vārds, uzvārds)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē
_____ prot. Nr. _____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)