Факултет техничких наука
Универзитета у Новом Саду

MapReduce и *HPCC* паралелна и дистрибуирана обрада података

Семинарски рад из предмета

Big Data у инфраструктурним систем

Ментор:
Проф. др. Александар Купусинац

Студент:
Илија Ракочевић E5 36/2023

Асистент:
Бојана Самарџић

Нови Сад, децембар, 2023.

Садржај

1. Увод

Многе организације имају велике количине података које су прикупљене и ускладиштене у виду скупова које треба обрађивати и анализирати како би се обезбедила пословна интелигенција (енгл. *business intelligence*), побољшали производи и услуге за клијенте, или за испуњавање других интерних захтева за обраду података[6]. На пример, нека интернет компанија треба да обрађује податке које прикупљају веб претраживачи, евиденцију података о кликовима као и друге информације које генеришу веб услуге[6]. Паралелизована технологија релационих база података није се показала исплативом нити обезбеђује високе перформансе потребне за анализирање великих количина података[6].

Као резултат тога, неколико организација је развило технологију која користи велике кластере за обезбеђивање рачунарских могућности високих перформанси (енгл. *high-performance computing*) за обраду и анализу масивних скупова података[6]. Кластери се могу састојати од стотина или чак хиљада јефтиних машина повезаних помоћу мрежа високог пропусног опсега[6]. Примери ове врсте кластер технологије укључују *Google MapReduce, Hadoop* и *LexisNexis HPCC* (*High-Performance Computing Cluster*) платформу који су и описани у овом раду[6].

2. High-Performance Computing

Високо-перформантно рачунарство (енгл. *High-Performance Computing*) користи се да опише рачунарска окружења које користе супер-рачунаре и рачунарске кластере како би се носиле са сложеним рачунарским захтевима, подржавале апликације са временски захтевним операцијама или захтевале обраду значајних количина података [6]. Супер-рачунари користе висок степен интерног паралелизма и обично користе специјализоване више-процесорске јединице са прилагођеним архитектурама меморије које су високо оптимизоване за нумеричке израчуне[6]. Супер-рачунари такође захтевају посебне технике паралелног програмирања како би искористили свој потенцијал у погледу перформанси[6].

Данашње мало скупље радне станице имају више рачунарске снаге од супер-рачунара који су постојали током раних деведесетих година[6]. То је довело до нове тенденције у дизајну супер-рачунара: коришћење кластера као независних процесора повезаних паралелно [6]. Многи рачунарски проблеми погодни су за паралелизацију; често се проблеми могу поделити тако да сваки независни процесорски чвор може радити на делу проблема паралелно, једноставно дељењем података које треба обрадити, а затим комбиновањем коначних резултата обраде за сваки део[6]. Овај тип паралелизма често се назива паралелизмом података, а апликације које то користе представљају потенцијално решење за захтеве обраде података у размерама петабајта[6].

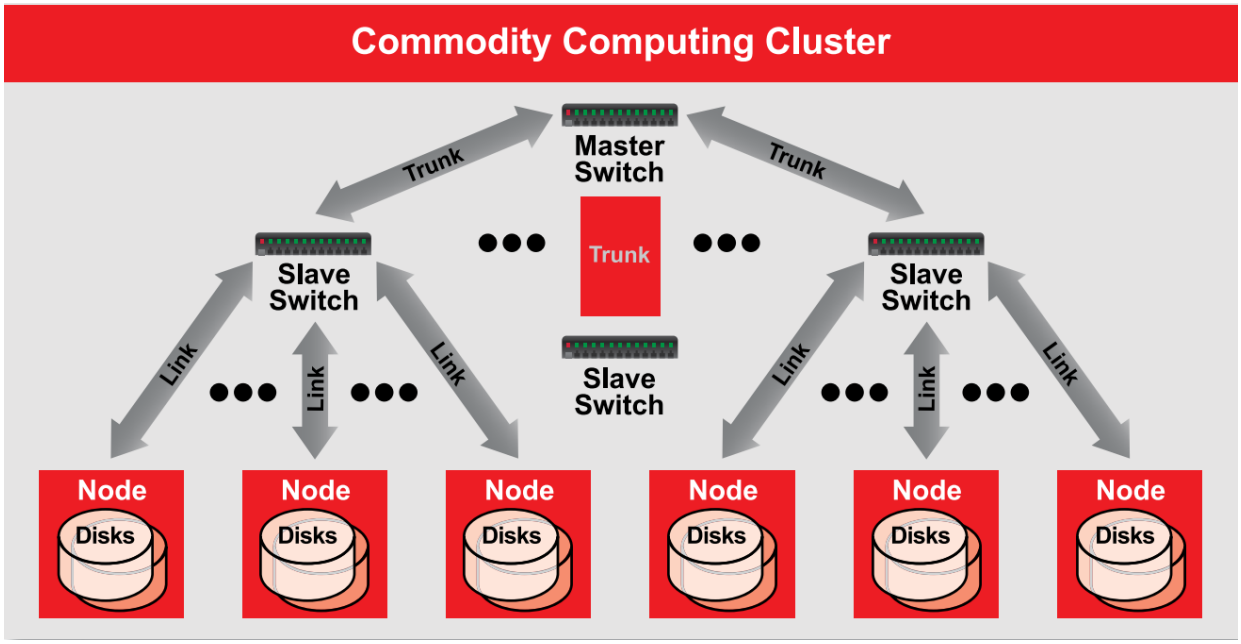
Паралелизам података може се дефинисати као обрада примењена независно на сваки податак скупа података, што омогућава да се степен паралелизма прилагоди обиму података[6]. Најважнији разлог за развој апликација које користе поменути паралелизам јесте потенцијал за скалабилне перформансе у високо-перформантном рачунарству, што може резултирати побољшањем перформанси од неколико редова величине[6]. Кључни проблеми су избор алгоритма, стратегија за декомпозицију података, равнотежа оптерећења на процесорским чворовима, комуникација између процесорских чворова и укупна тачност резултата[6].

2.1 Рачунарски кластери

Рачунарски кластер представља групу дељених појединачних рачунара, повезаних брзом комуникацијом у локалној мрежној топологији користећи технологију попут гигабитних мрежних прекидача (енгл. *switch*) или *InfiniBand*-ова, и укључује системски софтвер који пружа интегрисано окружење за паралелно процесирање апликација са способношћу дељења обраде између чворова у кластеру[6]. Конфигурације кластера не само да могу побољшати перформансе апликација које користе појединачни рачунари, већ пружају већу доступност и поузданост, и обично су много економичније од појединачних супер-рачунарских система са идентичним перформансама[6]. Кључ способности, перформанси и пропусности рачунарског кластера лежи у системсом софтверу и алатима који се користе за обезбеђивање окружења за извршавање паралелних послова[6]. Програмски језици са имплицитним могућностима паралелног процесирања и високим степеном оптимизације такође су потребни како би се обезбедили резултати високих перформанси, као и висока продуктивност програмера[6].

Кластери омогућавају подацима које користи апликација да буду подељени и обрађени међу доступним рачунарским ресурсима независно како би се постигле перформансе и

скалабилност спрам количине података[6]. Овај приступ паралелном процесирању често се назива *shared nothing* приступом, јер сваки чвор који се састоји од процесора, локалне меморије и диск ресурса не дели ништа са другим чворовима у кластеру, што је и приказано на слици 2.1[6]. Кластери су изузетно ефикасни када је релативно лако раздвојити проблем на неколико паралелних задатака и када нема зависности или комуникације између задатака осим укупног управљања задацима[6].



Слика 2.1 Структура рачунарског кластера

Високо-перформантни кластери се обично конфигуришу помоћу комерцијално доступних *PC* компоненти[6]. Сервери монтирани у *rack* ормане или *blade* сервери, сваки са локалном меморијом и диск складиштем, често се користе као процесорски чворови како би се омогућиле конфигурације високе густине са малим *footprint*-ом који олакшавају коришћење брзих комуникационих уређаја за повезивање чворова[6]. *Linux* се широко користи као оперативни систем[6]. Конфигурације кластера могу бити **симетричне** (сваки чвор може такође функционисати као засебан појединачни рачунар) или **асиметричне** (један рачунар функционише као главни чвор пружајући приступ корисницима и управљајући активностима других чворова), што је најчешћа архитектура[6]. Управљање кластером, безбедност и расподела радног оптерећења су мање проблематични, а оптималне перформансе обично су лакше постићи у асиметричним кластерима[6]. Хардвер који се користи у кластерима високих перформанси обично је хомоген, при чему сваки процесорски чвор садржи исте процесорске, меморијске и диск компоненте[6]. То омогућава системском софтверу да боље оптимизује радна оптерећења и пружа доследније перформансе за апликације паралелног процесирања[6]. У апликацији паралелног процесирања на кластеру где је радно оптерећење равномерно подељено, чвор који има спорији процесор или мање меморије ће заостајати за другим чворовима у извршавању свог дела апликације, што утиче на укупне перформансе[6].

2.2 *Data-intensive* апликације

Израз *data-intensive* користи се да опише рачунарске апликације које су *I/O bound* или имају потребу за обрадом великих количина података[6]. Такве апликације посвећују већи део свог времена за обраду улаза, излаза и кретање података[6]. Паралелно процесирање оваквих апликација обично укључује партиционисање или поддељење података у више сегмената који се могу независно обрадити користећи исти извршни програм паралелно на одговарајућој рачунарској платформи, а затим поновно састављање резултата како би се произвели комплетирани излазни подаци[6]. Што је већа укупна дистрибуција података, то је већа корист од паралелног процесирања података[6]. Захтеви за обрадом обично се линеарно скалирају према величини података и веома су подложни једноставној паралелизацији[6]. Постоји неколико важних карактеристика *data-intensive* система које их разликују од других облика рачунања,а то су[6]:

- Прво је **принцип размештаја података и програма/алгоритама за извођење рачунања**[6]. Да би се постигле високе перформансе у рачунању, важно је минимизирати кретање података[6]. Већина других типова рачунања користи податке смештене у одвојеном репозиторијуму или серверима и преносе податке у систем за обраду[6]. D*ata-intensive* систем обично користи дистрибуиране податке и дистрибуиране системе датотека у којима су подаци смештени широм кластера процесорских чворова, и уместо преноса података, програм или алгоритам се преноси на чворове са подацима који треба да буду обрађени[6]. Овај принцип - "Помери код ка подацима" - изузетно је ефикасан јер је величина програма обично мала у поређењу са великим скуповима података које обрађују *data-intensive* системи и резултира много мањим саобраћајем на мрежи, пошто подаци могу бити читани локално уместо преко мреже[6]. Ова карактеристика омогућава алгоритмима обраде да се извршавају на чворовима где се подаци налазе, смањујући системски *overhead* и повећавајући перформансе[6].
- Друга важна карактеристика је **коришћење програмског модела**[6]. Овакви системи обично користе приступ који је независан од машине, где се апликације изражавају помоћу операција над подацима које су на високом нивоу апстракције[6]. Систем за извршавање транспарентно контролише планирање, извршавање, равнотежу оптерећења, комуникацију и кретање програма и података широм дистрибуираног рачунарског кластера[6]. Апстракција приликом програмирања и језички алати омогућавају да се обрада изрази преко тока и трансформације података, укључујући нове језике за програмирање са фокусом на податке и дељење библиотека са заједничким алгоритмима манипулације подацима, као што је на пример сортирање[6]. Конвенционални системи за дистрибуирано рачунање обично користе моделе програмирања који зависе од машине, што може захтевати контролу програмера над нижим нивоима обраде и комуникације чворова користећи конвенционалне императивне језике програмирања и специјализоване софтверске пакете, што додаје комплексност задатку паралелног програмирања и смањује продуктивност програмера[6]. Модел програмирања зависан од машине такође захтева додатно време за подешавање и подложнији је за *single point of failure*[6].
- Трећа важна карактеристика система је **фокус на поузданост и доступност**[6]. Системи великих размера са стотинама или хиљадама процесорских чворова готово константно су подложни кваровима хардвера, грешкама у комуникацији и грешкама у софтверу[6]. *Data-intensive* системи су обично дизајнирани да буду отпорни на грешке (енгл. *fault-tolerant*)[6]. Ово укључује редундантне копије свих датотека података на диску, складиштење привремених резултата обраде на диску, аутоматско

откривање кварова чвора или обраде, и селективно поновно израчунавање резултата[6]. Кластер је обично у способности да настави са радом након квара неког чвора, уз аутоматско и транспарентно опорављање од недовршене обраде[6].

- Последња важна карактеристика је **скалабилност основне хардверске и софтверске архитектуре**[6]. Системи се обично могу скалирати линеарно да би се прилагодили готово било којој количини података или да би се задовољили временски зависни захтеви за перформансама, једноставним додавањем додатних процесорских чворова конфигурацији система како би се постигле стопе обраде од чак милијарди записа у секунди[6]. Број чворова и задатака обраде додељених за одређену апликацију може бити променљив или фиксан у зависности од хардвера, софтвера, комуникације и архитектуре дистрибуираног система датотека[6]. Ова скалабилност омогућава решавање рачунарских проблема који су некада сматрани неразрешивим због количине података која је била потребна или због времена обраде које је било потребно[6].

3. *MapReduce* алгоритам

Постоје различите архитектуре система које су имплементиране у *large-scale* апликацијама за анализу података, укључујући паралелне и дистрибуиране системе за управљање релационим базама података који су доступни за рад на *shared nothing* кластерима већ више од две деценије[6]. Овај приступ пружа предности када су подаци које систем користи структурирани и лако се уклапају у ограничења релационе базе података[6]. Међутим, проблем настаје приликом великог раста података који су у неструктурираном облику[6]. Било је потребно оформити нову парадигму за обраду података која би укључивала флексибилније моделе података[6]. Интернет компаније попут *Google*-а, *Yahoo*-а, *Facebook*-а и других захтевали су нови приступ обради како би се ефикасно носили са огромном количином веб података за апликације попут претраживача и друштвених мрежа[6]. Осим тога, многе владе и пословне организације биле су преплављене подацима које нису могле ефикасно обрадити, повезати и анализирати традиционалним рачунарским приступима[6].

Неколико решења се појавило, укључујући *MapReduce* архитектуру коју је први пут развио *Google*, а сада је доступна у *open-source* имплементацији названој *Hadoop*, коју користе *Yahoo, Facebook* и друге компаније[6]. *Google MapReduce* је пример основне архитектуре система дизајнираног за обраду и анализу великих скупова података на кластерима рачунара опште намене и успешно се користи од стране *Google*-а у многим апликацијама за обраду великих количина необрађених веб података[6]. *LexisNexis,* такође је развио и имплементирао скалабилну платформу названу *HPCC*, представљену у секцији 4, која нуди значајно више могућности од *MapReduce* алгоритма и већ неколико година се користи од стране *LexisNexis*-а и других комерцијалних и владиних организација за обраду веома великих количина структурисаних и неструктурисаних података[6].

MapReduce програмски модел омогућава паралелно груписање агрегација преко кластера машина[6]. Обрада се аутоматски паралелизује од стране система на кластеру, који се брине о детаљима попут партиционисања улазних података, планирања и извршавања задатака широм кластера, и управљања комуникацијом између чворова, омогућавајући програмерима без искуства у паралелном програмирању да користе велико паралелно окружење за обраду[6]. За сложеније поступке обраде података, више позива алгоритма мора бити повезано заједно у секвенци[6].

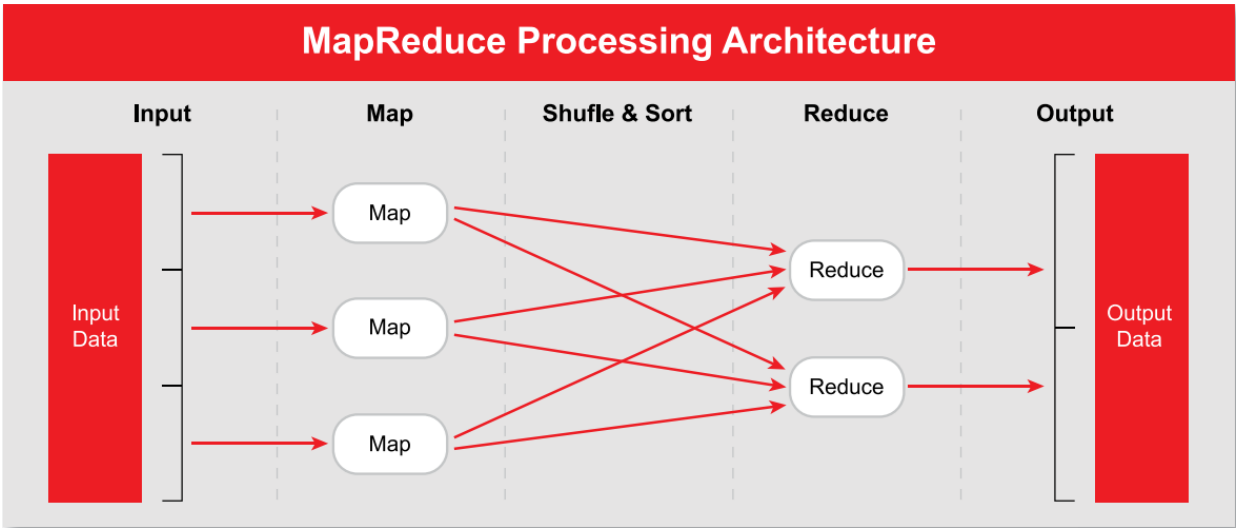
3.1 Опис алгоритма

Корисник *MapReduce* библиотеке, рачунски део алгоритма изражава преко две функције: *Map* и *Reduce*[1], приказаних на слици 3.1.

Map функција, написана од стране корисника, користи улазне парове и производи скуп међурезултата парова кључ-вредност (*key-value*)[1]. Након тога библиотека групише све међурезултате по истој вредности кључа и прослеђује их *Reduce* функцији[1].

Reduce функција, такође написана од стране корисника, прихвата прослеђене кључеве заједно са скупом вредности које су им додељене[1]. У овом тренутку се врши спајање вредности по кључевима како би се потенцијално смањили скупови[1]. Ова фунцкија приликом позива, углавном као излаз даје једну вредност или је без повратне вредности[1].

Међурезултати парова кључ-вредност су спроведени до *Reduce* функције коришћењем итератора[1]. На тај начин се омогућава управљање листама вредности које су превише велике за складиштење у меморију[1].



Слика 3.1 Шема рада MapReduce-а

3.2 Примери примене

3.2.1 Пример 1

Приликом посматрања проблема бројања појављивања сваке речи у великој колекцији докумената, корисник ће написати код који је сличан псеудо коду приказаном на слици 3.2[1].

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Слика 3.2 Псеудо код за проблем бројања појављивања сваке речи

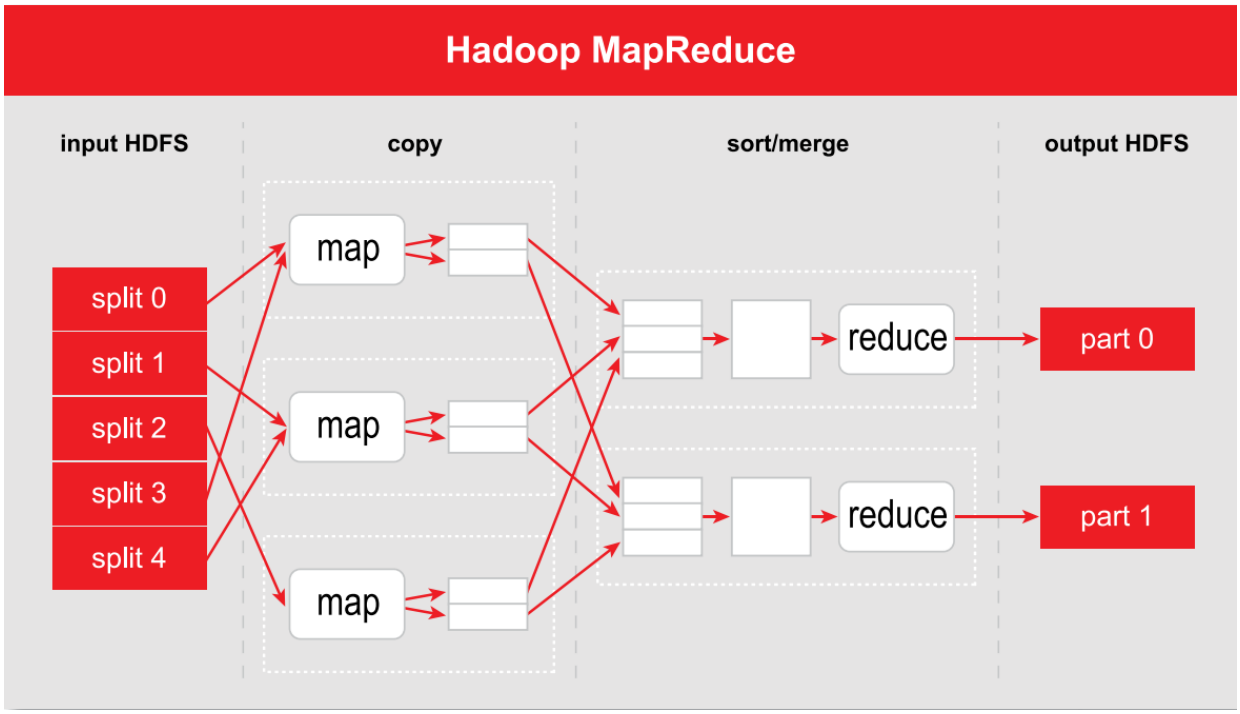
Map функција емитује сваку реч и припадајући број појављивања (у овом примеру само једно понављање)[1]. *Reduce* функција затим сабира све бројеве емитоване за одређену реч[1]. Поред тога, корисник пише код како би попунио објекат спецификације *MapReduce* модела са именима улазних и излазних датотека, као и опционе параметре за подешавање[1].

3.2.2 Пример 2

Пример бројања фреквентности приступа одређеном УРЛ-у се може применити на следећи начин[1]. *Map* функција процесуира логове захтева ка веб страницама и као излаз даје пар (УРЛ, 1); док *Reduce* функција сабира све вредности за исти УРЛ и емитује пар (УРЛ, укупна сума)[1].

3.3 *Hadoop* софтвер

Hadoop је пројекат отвореног кода који је спонзорисан од стране *The Apache Software Foundation* (http://www.apache.org)[2, 5]. Покренут је како би се створила *open-source* имплементација *MapReduce* архитектуре[2, 5]. *Hadoop MapReduce* архитектура, приказана на слици 3.3 функционално је слична *Google* имплементацији, при чему је основни програмски језик за *Hadoop Јава* уместо *C++*[2, 5]. Имплементација је намењена извршавању на кластерима процесора опште намене, користећи *Linux* као оперативни систем[2, 5].



Слика 3.3 Приказ Hadoop MapReduce-а

Hadoop имплементира окружење и радни оквир за распоређивање и извршавање дистрибуиране обраде података за *MapReduce* послове[2, 5]. *MapReduce* посао је јединица рада која се састоји од улазних података, повезаних *Map* и *Reduce* програма, и кориснички одређених конфигурационих информација[2, 5]. *Hadoop* радни оквир користи архитектуру господар-роб (енгл. *master-slave*), где постоји један главни сервер назван *jobtracker* и помоћни сервери названи *tasktrackers*, при чему постоји само један *tasktracker* по чвору у кластеру[5, 6]. *Hadoop* укључује дистрибуирани систем датотека (енгл. *Hadoop Distributed File System - HDFS*), који је аналоган гугл систему датотека (енгл. *Google File System - GFS*) у *Google*-овој имплементацији *MapReduce*-а[6]. *HDFS* такође следи архитектуру господар-роб која се састоји од једног главног сервера који управља дистрибуираним именником система датотека (енгл. *Distributed Filesystem*

Namespace) и регулише приступ датотекама од стране клијента, названог *Namenode*[6]. Осим тога, постоји више чворова названих *Datanode*, један по чвору у кластеру, који управљају дисковним складиштем које је припојено чворовима[6].

Hadoop окружење за извршавање послова подржава додатне способности дистрибуиране обраде података које су дизајниране за рад користећи *Hadoop MapReduce* архитектуру, као што је *Pig* систем[5, 6]. *Pig* укључује језик и окружење које је оригинално развијено у *Yahoo*! да пружи специфичну нотацију језика за апликације анализе података и побољша продуктивност програмера, смањујући циклусе развоја приликом коришћења *Hadoop MapReduce* окружења[5, 6]. Синтакса *Pig* језика је приказана на слици 3.4.

Sample Pig Latin Program

```
visits      = load '/data/visits' as (user, url, time);
gVisits     = group visits by url;
visitCounts = foreach gVisits generate url, count(urlVisits);

urlInfo     = load '/data/urlInfo' as (url, category, pRank);

visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls     = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

Слика 3.4 Приказ Pig синтаксе

Језгро *Hadoop*-а чине *HDFS* и *MapReduce*[4]. Након развијеног језгра, велики број софтвера је изграђен тако да функционише користећи ова два елемента[4]. Неки софтвери омогућавају лакши увоз података на *Hadoop* платформу, док је велики број софтвера развијен како би олакшало његово коришћење[4].

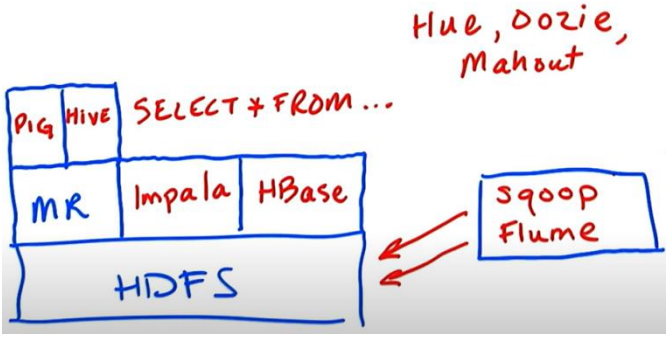
Pig и *Hive* су популарни код корисника који не познају програмске језике, а желе да пишу упите ка својим подацима, јер не захтевају писање *MapReduce* кода[4]. Наредбе ка подацима се углавном своде на *SQL* наредбе[4]. *Hive* интерпретер преводи упите у *MapReduce* код и бива покренут на кластеру[4]. *Pig* омогућава писање захтева за анализу података у практично једноставном језику[4]. Такође, и овај код се преведи у *MapReduce* код и извршава на кластеру[4]. Коришћење ова два приступа изискује више времена, због додатног превођења[4].

Потреба за додатним временом се може избећи коришћењем *Impala* решења, које користи *SQL* упите и директно приступа подацима на *HDFS*-у[4].

Sqoop узима податке из традиционалних релационих база података (нпр. *Microsoft SQL Server*) и убацује их у *HDFS*; на тај начин да могу бити процесуирани заједно са осталим подацима на кластеру[4].

Flume узима податке који су генерисани у неким екстерним системима и доставља их у кластер[4].

HBase је *real-time* база података, која је изграђена на самом *HDFS*-у[4]. Структура екосистема је приказана на слици 3.5[4].



Слика 3.5 Hadoop екосистем

3.4 Тренутна ограничења *MapReduce* алгоритма.

Иако *MapReduce* модел и програмерска апстракција пружају основне функционалности за многе операције обраде података, корисници су ограничени његовом крутом структуром и присиљени су прилагодити своје апликације моделу како би постигли паралелизам[3, 6]. Ово може захтевати имплементацију вишеструких *MapReduce* секвенци за захтевније обраде које можда захтевају извршавање више секвенцијалних операција или операција као што је спајање више улазних датотека[3, 6]. На тај начин се може додати значајан *overhead* управљања задацима, целокупном времену обраде, као и ограничити могућности оптимизације обраде различитим стратегијама извршавања[3, 6]. Осим тога, многе операције обраде података не уклапају се природно у модел груписања по агрегацији, користећи појединачне парове кључ-вредност које модел захтева[3, 6]. Чак и једноставне операције попут пројекције и селекције морају бити уклопљене у овај модел, а корисници морају обезбедити прилагођене *Map* и *Reduce* функције за све апликације, што је подложније грешкама и ограничава поновну употребу [3, 6]. С обзиром да *Map* и *Reduce* функције морају бити обезбеђене за сваки корак, неспособност за глобалном оптимизацијом извршавања сложених секвенци обраде података може резултирати значајно смањеним перформансама[3, 6].

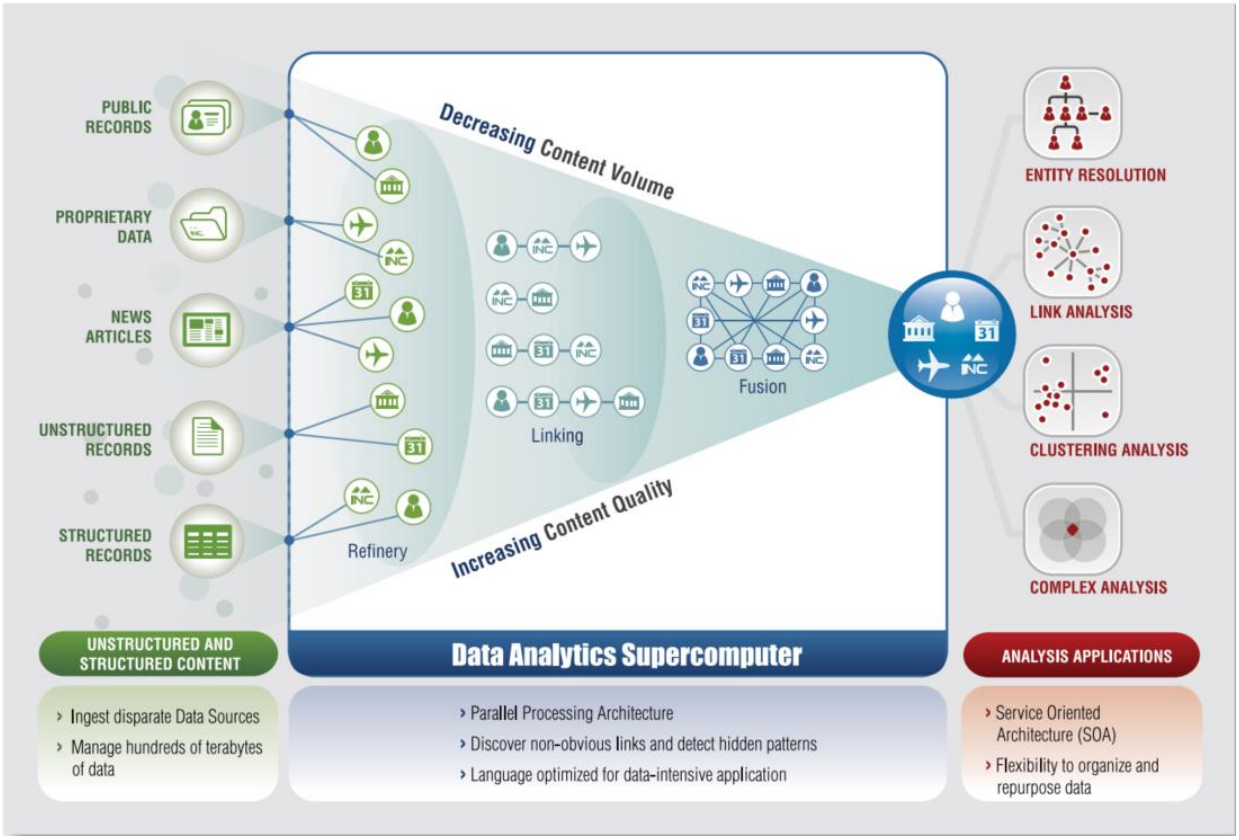
Google са својим језиком *Sawzall* и *Yahoo* са својим *Pig* системом и језиком за *Hadoop* адресирају нека од ограничења *MapReduce* модела пружањем екстерног језика оријентисаног ка обради података, који преводи језичке наредбе у *MapReduce* секвенце обраде[6]. Ови језици пружају много стандардних оператора обраде података тако да корисници не морају да имплементирају прилагођене *Map* и *Reduce* функције, побољшају реискористивост и пруже неку оптимизацију за извршавање послова[6]. Међутим, ови језици су екстерно имплементирани и извршавају се на клијентским системима и нису интегрални део *MapReduce* архитектуре, али и даље се ослањају на исту инфраструктуру и ограничени модел извршавања који пружа *MapReduce*[6].

MapReduce модел је дизајниран за рад у паралелном окружењу обраде у блоковима, што је корисно за извођење *ETL*(*Extract, Transform, Load*) послова на великим скуповима података који се морају трансформисати за неке друге сврхе[6]. Систем је такође користан за извршавање упита у блоковима (енгл. *batch*) који обављају агрегационе операције или сложене аналитичке задатке на великим скуповима података, а посебно на неструктурираним подацима јер нема потребе за изградњом индекса или учитавањем у *RDBMS* (*Relational Database Management System*)[6]. Међутим, за ефикасно *online* претраживање великих скупова података који морају подржавати велики број корисника или пружати брзе одговоре са насумичним приступом структуираним подацима и подржавати апликације складишта података као што је то случај са паралелним *DBMS* системима, потребне су друге платформе у оквиру *MapReduce* окружења[6]. Google је адресирао ову потребу додавањем *BigTable*, док *Hadoop* користи већ поменуте *HBase* и

Hive[6]. Ове компоненте раде суштински као додаци *MapReduce* архитектури користећи основне системе складиштења датотека и *MapReduce* обраду, али иначе раде као независне неинтегрисане апликације[6]. Бољи приступ био би интегрисано окружење система које се одлично сналази како у *ETL* задацима тако и у сложеним анализама, у ефикасном претраживању великих скупова података користећи заједнички *data-centric* језик паралелне обраде података. Систем платформе *LexisNexis HPCC* је управо дизајниран за те сврхе.

4. *HPCC* платформа

LexisNexis, лидер у индустрији која је оријентисана на податке, агрегације података и информационих услуга, самостално је развио и имплементирао решење за рачунарску обраду података под називом *HPCC* (*High-Performance Computing Cluster*), познато и као Супер-рачунар за Аналитику Података (енгл. *Data Analytics Supercomputer*)[6]. Визија *LexisNexis*-а за ову рачунарску платформу приказана је на слици 4.1[6].



Слика 4.1 LexisNexis визија за HPCC платформу

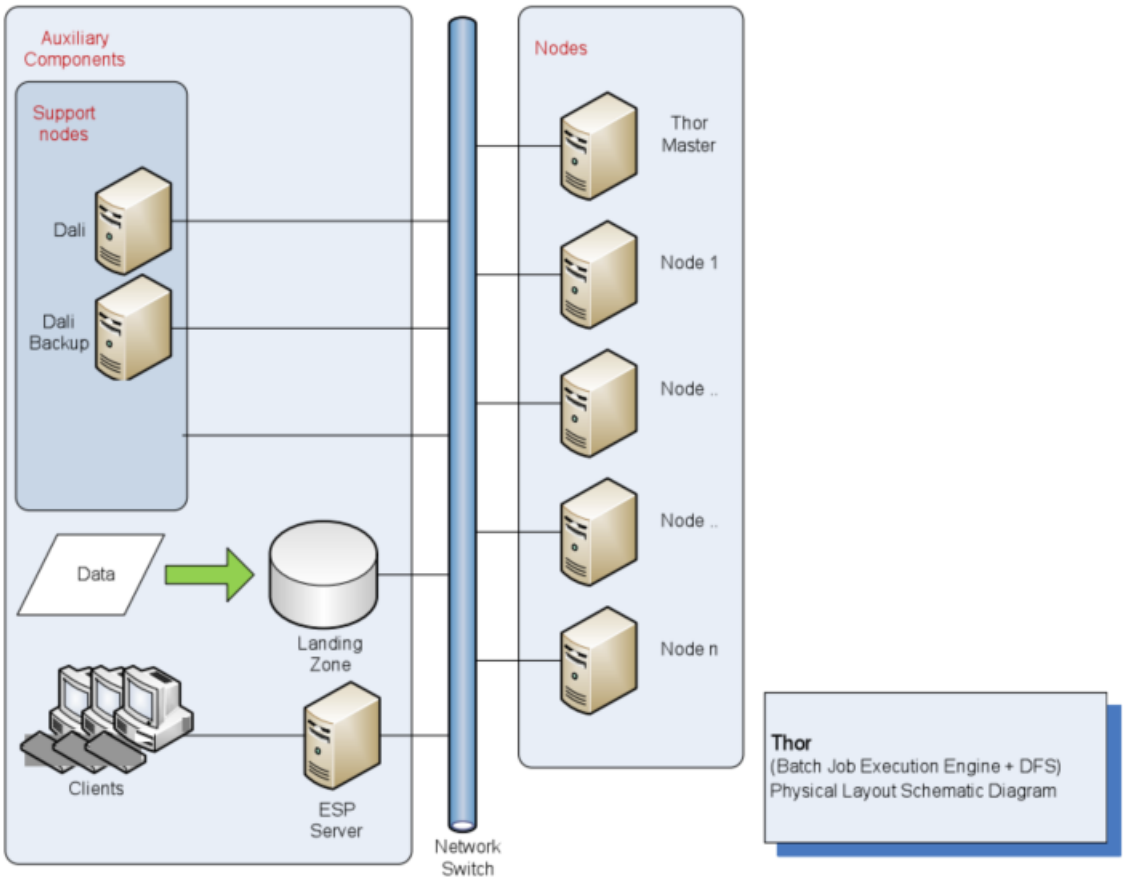
Развој ове рачунарске платформе од стране *Seisint, Inc.* (затражено од стране *LexisNexis*-а 2004. године) започео је 1999. године, а апликације су биле у продукцији крајем 2000. године. *LexisNexis*-ов приступ такође користи комерцијалне кластере хардвера који покрећу *Linux* оперативни систем, слично кластеру приказаном на слици 3.1[6]. Прилагођени системски софтвер и компоненте посредника (енгл. *middleware*) развијене су и постављене на основни *Linux* оперативни систем како би обезбедили извршно окружење и подршку дистрибуираном систему датотека потребном за *data-intensive* обраду[6]. Приступ дизајну укључивао је и дефинисање новог програмског језика високог нивоа за паралелно процесирање података названог *ECL* (*Enterprise Data Control Language*)[6].

4.1. Типови кластера

Развојни тим *LexisNexis*-а препознао је да испуњавање свих захтева апликација за *data-centric* обраду на оптималан начин захтева дизајн и имплементацију два различита кластер окружења обраде, од којих сваки може бити независно оптимизован за своју сврху паралелног процесирања података[6].

4.1.1 *Thor* платформа

Прва од ових платформи назива се *Data Refinery*, чија је основна сврха општа обрада масивних обима сирових података било ког типа у било које сврхе, али се обично користи за чишћење и хигијену података, *ETL* процесирање сирових података, повезивање записа и разрешавање ентитета, *large-scale adhoc* сложених аналитика и креирање података са кључевима и индексима ради подршке захтевним структурираним упитима и апликацијама за складиштење података[6]. *Data Refinery* се такође назива *Thor*, по референци на митолошког нордијског бога грома с великим чекићем симбоше дробљење великих количина сирових података у корисне информације[6]. *Thor* кластер је сличан платформама *Google* и *Hadoop MapReduce* по својој функцији, извршном окружењу, систему датотека и способностима, али нуди значајно већу перформантност у еквивалентним конфигурацијама[6].



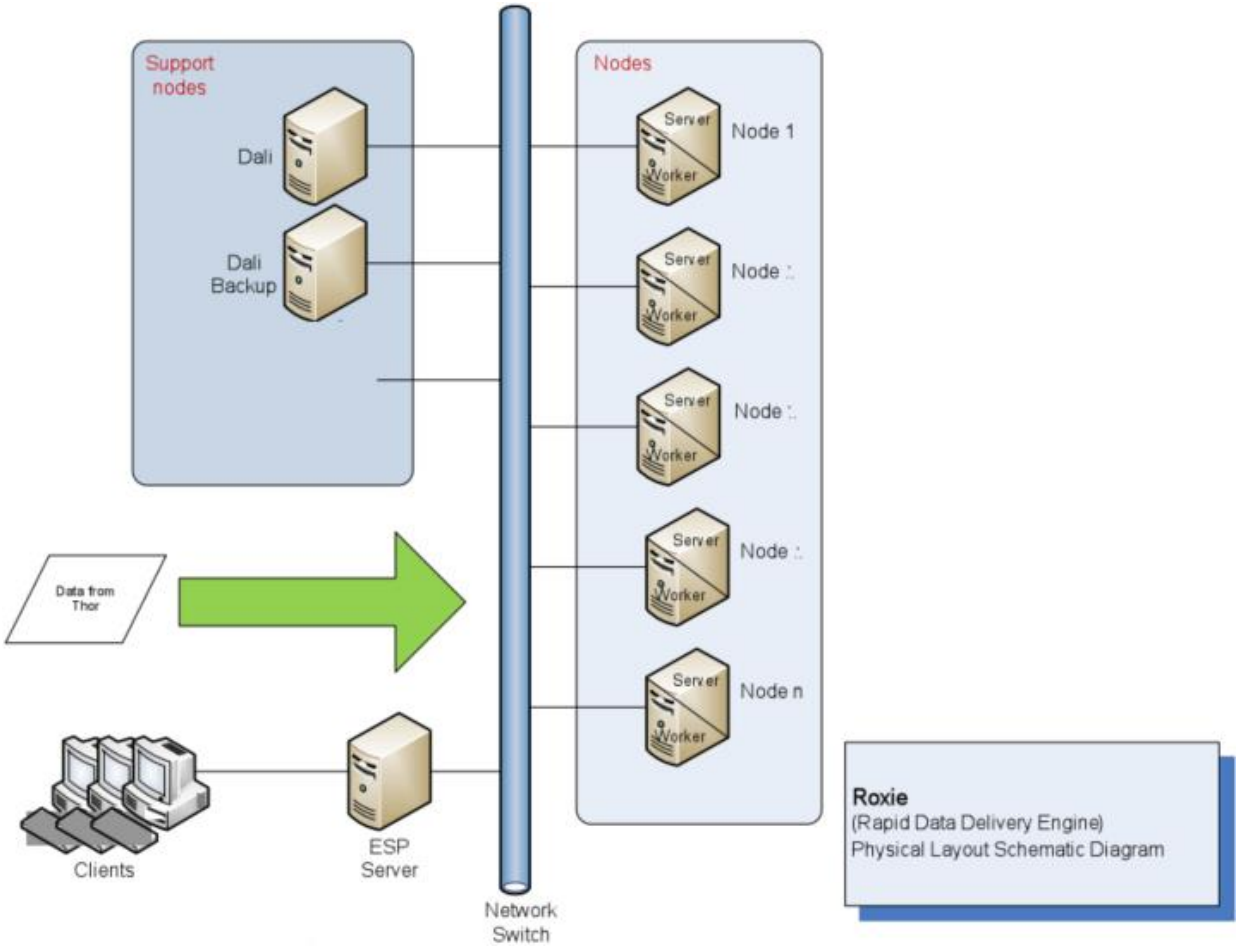
Слика 4.2 Thor платформа

Слика 4.2. приказује физичку реализацију *Thor* процесорског кластера који функционише као извршни мотор за обраду низова послова за *data-centric* скалабилне апликације[6].

Осим *Thor* главних и помоћних чворова (енгл. *master-slave*), потребне су додатне помоћне и заједничке компоненте како би *HPCC* окружење било имплементирано у потпуности за обраду[6]. Тачан број чворова потребних за помоћне компоненте одређује се током процеса конфигурације[6].

4.1.2 *Roxie* платформа

Друга платформа за паралелно процесирање података дизајнирана и имплементирана од стране *LexisNexis*-а назива се *Rapid Data Delivery Engine*[6]. Ова платформа је дизајнирана као *online* платформа за структуриране упите и анализе високих перформанси или складиштење података које пружа захтеве за паралелним приступом подацима. Користи се за *online* апликације путем веб сервисних интерфејса који подржавају хиљаде истовремених упита и корисника са брзином одговора мереном у милисекундама[6]. *Rapid Data Delivery Engine* се такође назива *Roxie*, акроним за *Rapid Online XML Inquiry Engine*[6]. *Roxie* користи посебан дистрибуирани индексирани систем датотека за пружање паралелног процесирања упита[6]. *Roxie* кластер је сличан по функцији и могућностима *Hadoop*-у уколико му се додају могућности *HBase*-а и *Hive*-а[6]. Важно је напоменути да, како *Thor* тако и *Roxie* кластери користе исти *ECL* програмски језик за имплементацију апликација, повећавајући континуитет и продуктивност програмера[6].



Слика 4.3 Roxie платформа

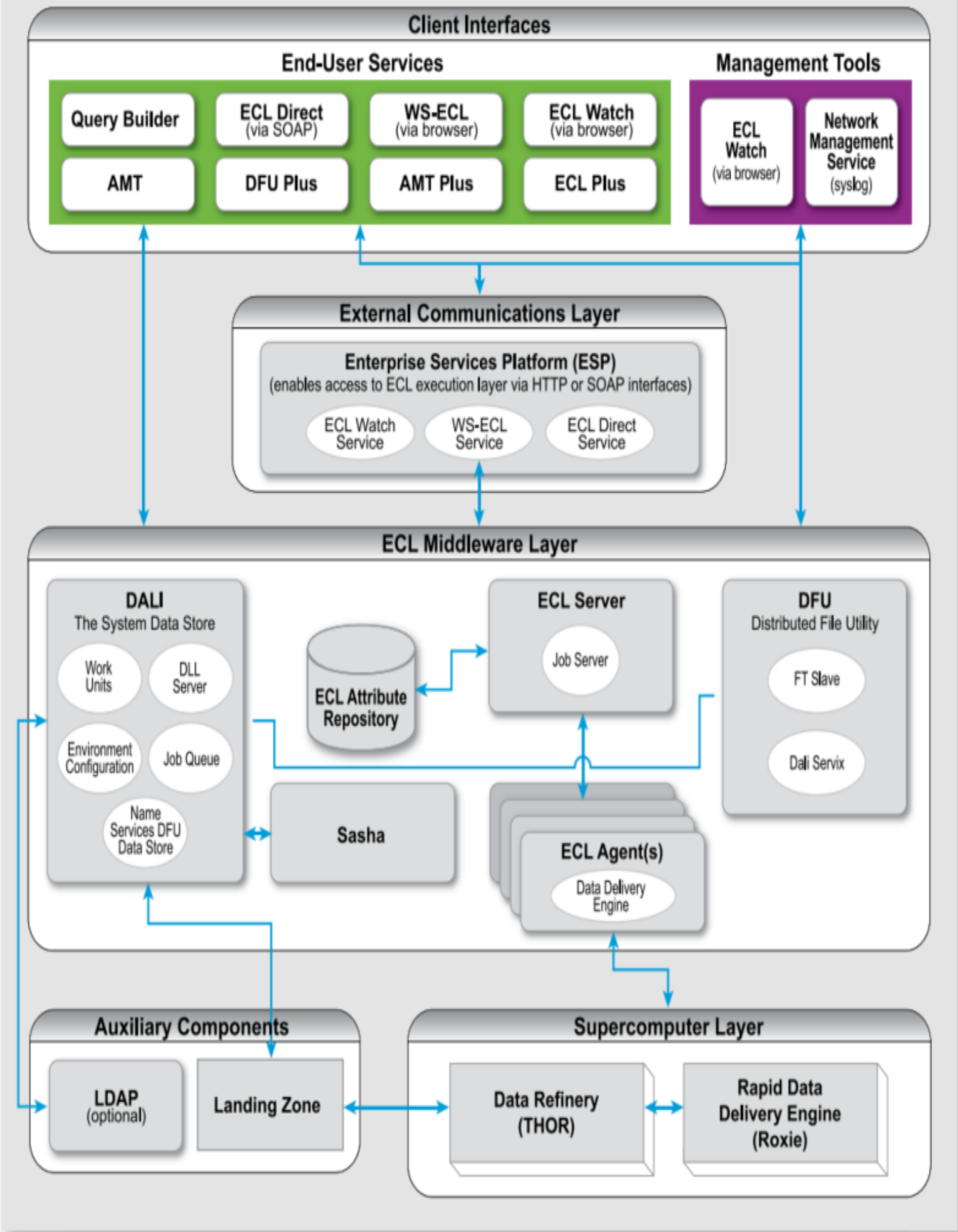
Слика 4.3 приказује физичку реализацију *Roxie* процесорског кластера. *Roxie* кластер

укључује више чворова са сервером и радним процесима за обраду упита; додатну помоћну компоненту названу *ESP* сервер која пружа интерфејсе за спољни приступ кластеру од стране клијената; и додатне заједничке компоненте које се деле са *Thor* кластером у *HPCC* окружењу[6]. Иако се *Thor* процесорски кластер може имплементирати и користити без *Roxie* кластера, *HPCC* окружење које укључује *Roxie* кластер мора укључивати *Thor* кластер[6]. *Thor* је потребан за изградњу дистрибуираних индекс датотека које користи *Roxie* и за развој *online* упита који ће бити имплементирани заједно са индекс датотекама у *Roxie* кластеру[6].

Имплементација два типа паралелних платформи за обраду података (*Thor* и *Roxie*) у *HPCC* процесном окружењу, које задовољава различите потребе за обрадом података, омогућава оптимизацију и подешавање ових платформи за њихове специфичне намене како би се корисницима пружио највиши ниво системских перформанси[6]. Ово је значајна предност у поређењу с *Hadoop*-ом, где се архитектура *MapReduce*-а мора надоградити додатним системима као што су *HBase*, *Hive* и *Pig*, који имају различите циљеве и захтеве за обрадом, и не увек лако одговарају парадигмама *MapReduce*-а[6]. Осим тога, *LexisNexis HPCC* приступ укључује појам процесорског окружења које може интегрисати *Thor* и *Roxie* кластере по потреби како би задовољиле све процесорске потребе организације[6]. Као резултат тога, скалабилност се може дефинисати не само у погледу броја чворова у кластеру, већ и у погледу броја кластера и њихове међусобне интеграције како би се испунили циљеви перформанси система и захтеви корисника[6]. Ово пружа значајну флексибилност у поређењу с *Hadoop* кластерима који често функционишу као потпуно независне јединке обраде[6].

Архитектура *HPCC* система укључује *Thor* и *Roxie*, заједничке помоћне компоненте, слој за спољне комуникације, корисничке интерфејсе који пружају услуге крајњим корисницима и алатке за управљање системом, као и помоћне компоненте за подршку праћењу и олакшавању учитавања и чувања података датотечког система са спољних извора[6]. Комплетна архитектура система приказана је на слици 8[6].

HPCC System Architecture



5. Закључак

Као резултат наставка експлозије информација, многе организације се суочавају са неспособношћу да обраде ове информације и ефикасно их користе. Високо-перформантно *data-intensive* рачунарство уз помоћ рачунарских кластера опште намене представља нови приступ који може решити поменути проблем и омогућити владиним и комерцијалним организацијама, као и истраживачким окружењима, да обрађују масивне количине података и имплементирају нове апликације које су раније сматране непрактичним или неизводљивим. Технолошка решења попут *Hadoop MapReduce* и *HPCC* платформе од стране *LexisNexis* сада су доступна и пружају могућност обраде података паралелно на јефтиним рачунарским кластерима опште намене.

Прикладност платформе и архитектуре за организацију и њене захтеве за апликацијама може се одредити тек након пажљиве евалуације доступних алтернатива. Многе организације су прихватиле *open-sourc*е платформе попут *Hadoop*-а, док друге преферирају комерцијално развијене и подржане платформе од стране установљених лидера у индустрији, као што је *HPCC*.

Постоји много компромиса приликом доношења исправне одлуке у вези са избором нове архитектуре рачунарских система, а често је најбољи приступ спровести специфично тестирање перформанси са апликацијом клијента како би се утврдила укупна ефикасност и перформантност система. Релативне карактеристике трошка и перформанси система, узимајући у обзир одговарајућу флексибилност, скалабилност, заузеће простора и факторе потрошње енергије који утичу на укупни трошак (енгл. *The total cost of ownership - TCO*).

6. Литература

[1] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters, 2004.

[2] „What is Apache hadoop?“, https://cloud.google.com/learn [Online], Доступно на: https://cloud.google.com/learn/what-is-hadoop. [Приступано: 12.12.2023].

[3] Дипломски рад од Каролине Воларић на тему: „Примена MapReduce алгоритма на анализу низова текстуалних података“, https://repozitorij.pmf.unizg.hr [Online], Доступно на: https://repozitorij.pmf.unizg.hr/islandora/object/pmf:5576/datastream/PDF/download. [Приступано: 12.12.2023].

[4] „Intro to Hadoop and MapReduce”, www.youtube.com [Online], Доступно на: https://www.youtube.com/playlist?list=PLAwxTw4SYaPkXJ6LAV96gH8yxIfGaN3H- [Приступано: 12.12.2023].

[5] „Hadoop: The Definitive Guide”, www.isical.ac.in [Online], Доступно на: https://www.isical.ac.in/~acmsc/WBDA2015/slides/hg/Oreilly.Hadoop.The.Definitive.Guide.3rd.Edition.Jan.2012 [Приступано: 21.12.2023].

[6] „HPCC Systems: Introduction to HPCC (High-Performance Computing Cluster)”, docs.huihoo.com [Online], Доступно на: https://docs.huihoo.com/hpcc/Introduction-to-HPCC [Приступано: 23.12.2023].