ARHITEKTURA IN IZBOR PROGRAMSKE TER STROJNE OPREME ZA POGANJANJE RAY-TRACING ALGORITMOV

Tehnično poročilo

Revizija:	1.0
Projekt:	ARRS-ART
Avtorji:	Vito Čuček, Marjan Šterk, Matic Herman, Gregor Pipan
Datum:	7. marec 2017

Kazalo

Kazalo slik

Slovar izrazov

Hypervisor	Program za upravljanje in izgradnjo virtualnih naprav.
GPU	Grafični procesor, sestavljen iz večjega števila manjših procesnih enot, specializiranih za aritmetične operacije.
CUDA	Programski jezik prilagojen za programiranje NVIDIA grafičnih procesorjev.
Instanca	Manifestacija programske komponente ali virtualnega stroja na podlagi programske kode ali slike.
Node	Skupek virtualnih ali fizičnih naprav, za delovanje neodvisen od ostalih, v omrežju.
CPU	Centralna procesorska enota.
OptiX	CUDA knjižnica za lažji razvoj algoritmov sledenja žarkom.
API	Aplikacijski vmesnik.
Komponenta	Logično ločena programska funkcionalnost večjega sistema.

1 Uvod

Tehnično poročilo je izdelano kot rezultat programskega sklopa 2.1 manjšega aplikativnega raziskovalnega projekta Advanced Ray-tracing Technics in Radio Environment (krajše ART), pridobljenega na razpisu Javne agencije za raziskovalno dejavnost Republike Slovenije (krajše ARRS). Namen projekta je karakterizirati radijsko okolje in izboljšati radijsko lokalizacijo s pomočjo naprednih tehnik sledenja žarkom.

V naslednjih poglavjih bomo opisali primernost oblačne infrastrukture ali krajše IaaS (ang. Infrastructure as a Service) za izvajanje simulacij, ki temeljijo na pristopu sledenja žarkov (ang. Ray-Tracing), in predlagali programsko arhitekturo za vzporedno procesiranje.

Trenutni algoritem, razvit na Inštitutu Jožef Štefan, izrablja tehniko grafičnih cevovodov in CUDA procesorje, za katere je znano, da imajo pri izračunavanju medsebojno neodvisnih aritmetičnih operacijah za nekaj redov večjo računsko moč od centralnega procesorja. Pri sledenju žarkov je gledano iz performančnega stališča ključnega pomena logika izgradnje drevesne strukture poligonov in posledično iskanje presečišč med žarki in okolico. OptiX knjižnica podjetja NVidia, katero izrablja tudi ART projekt, omogoča relativno hiter in optimiziran način izdelave simulacijskih algoritmov, zasnovanih na metodi sledenja žarkov.

V okviru programskega sklopa 2.1 smo preučili primerne programske in strojne komponente, arhitekturo, izdelal OS neodvisen build sistem in predrugačili izvorno kodo aplikacije ART. Slednje je bilo potrebno, da lahko posamezne dele aplikacije uporabljamo ločeno in da je aplikacija združljiva z operacijskim sistemom Linux, ki se pretežno uporablja v oblačnih storitvah.

2 Oblačna infrastruktura

Oblačna infrastruktura nam omogoča streženje virtualnih sredstev na zahtevo. Predstavlja osnovni nivo oblačnih storitev, ki mu pravimo infrastruktura kot storitev ali IaaS (ang. Infrastructure as a Service). Storitve na višjih nivojih uporabljajo IaaS za delovanje in streženje aplikacij. Te prevzemajo vloge končnih storitev SaaS (ang. Software as a Service) ali pa nudijo programsko okolje PaaS (ang. Platform as a Service).

Od ponudnika oblačne storitve ali od sistema, ki nudi oblačno storitev, je odvisno kateri nivo uporabnikom nudi. V našem primeru se osredotočamo samo na sisteme ali ponudnike, ki nudijo oblačne storitve na strojnem nivoju (IaaS), zaradi večje kompatibilnosti med sistemi in fleksibilnosti pri razvoju. IaaS oblacne storitve so najbolj primerne za izdelavo HPC (ang. High Preformance Cluster) sistema, zaradi moznosti spreminjanja in nadzora delovanja na nivoju operacijskega sistema.

2.1 Horizontalna skalabilnost

Hitrost izračuna propagacije radijskih valov skozi prostor in poustvarjanje raznih pojavov je močno odvisno od strojne infrastrukture, programske implementacije in zahtevane natančnosti. Oblačna infrastruktura s pravo programsko implementacijo in algoritmom, prilagojenim za vzporedno izvajanje, nima navzgor omejene skupne procesorske moči. Takšen sklop strojne in programske opreme razumemo kot horizontalno skalabilni sistem.

2.2 Infrastruktura kot storitev

Za razvoj, testiranje in streženje skalabilne programske storitve zadošča običajna gruča medsebojno neodvisnih strojnih enot s preprosto zvezdno topologijo omrežja. Še lažji razvoj in testiranje pa omogočijo oblačne storitve, ker nudijo virtualizirane strojne komponente na zahtevo, brez fizičnega posredovanja. Poleg tega omogočajo enostavno deljenje strojnih sredstev med posameznimi aplikacijami za dosego boljših izkoristkov. Aplikacija se s pomočjo tako imenovane elastične infrastrukture samodejno širi ali krči glede na zahteve.

2.3 OpenStack platforma kot računalništvo v oblaku

Trenutno na tržišču obstaja mnogo ponudnikov oblačnih storitev (AWS, Google Cloud, Azure, Heroku, Linode...), od katerih so vsa plačljiva in ne omogočajo uporabe lastne fizične infrastrukture. Odločilismo se, da bomo za

potrebe testiranja uporabili odprtokodno oblačno platformo OpenStack, ki temelji na IaaS nivoju. Nameščena je na naši lastni infrastruktuti in jo že nekaj let uporabljamo za potrebe raziskovalnih in razvojnih projektov. Implementacija storitve projekta ART ne postavlja v ospredje nobene oblačne infrastrukture ali tipa virtualizacije, dokler virtualizirano okolje podpira knjižnico CUDA 7 in operacijski sistem Linux. Opisali bomo splošno arhitekturo platforme ter postopke in načine virtualizacije grafičnih procesorjev in tako pripomogli pri celostnem razumevanju sistema oblačnih storitev.

OpenStack je modularen sistem, prvenstveno namenjen za virtualizacijo in streženje infrastrukture (IaaS-Infrastructure as a service).

Glavne komponente so Nova, Cinder, Neutron, Glance, Swift in Ironic. Vsaka od naštetih komponent nudi API, preko katerega Horizon in Heat upravljata celotni sistem. Horizon nudi uporabniški vmesnik, preko katerega lahko dostopamo, spreminjamo ali ustvarjamo nove virtualne naprave, Heat pa visokonivojske ukaze razdeli na manjše opracije in jih posreduje posameznim namenskim komponentam. Na ta način ureja življenski cikelj virtualnih naprav ali z drugimi besedami orkestrira infrastrukturo in aplikacije v okolju OpenStack.

Nova predstavlja glavno vstopno točko za izgradjno virtualnih računskih naprav, ker služi kot abstrakcija raznih virtualizacijskih tehnik, katere poganja hypervisor. Podprte tehnike virtualizacije so KVM, VMware, Xen, LXC, Hyper-V, Bare-metal. Primerne načine virtualizacij, za porebe projekta ART, bomo predtavili v poglavju 2.5.

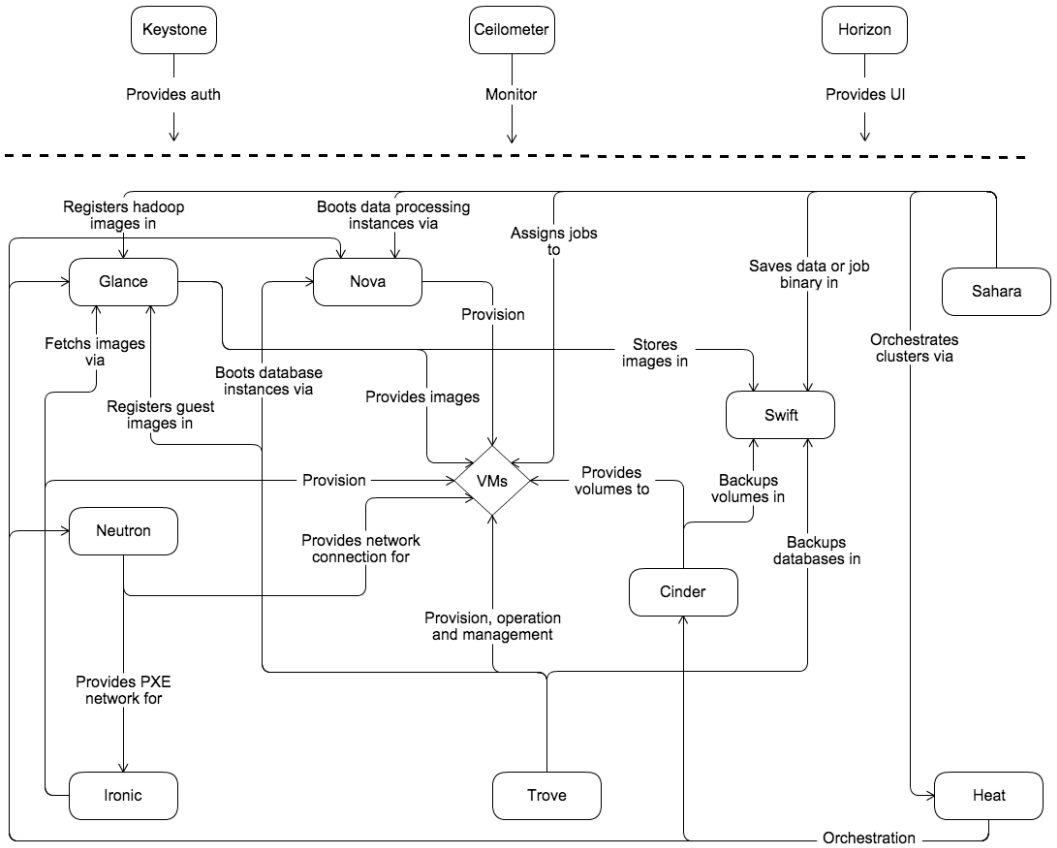
Cinder služi za upravljanje in virtualizacijo bločnih naprav (ang. block device), katere preko datotečnih gonilnikov uporablja Nova. Uporabnik lahko tako virtualizirani računski napravi dodeli datotečni sistem.

Glance je namenjen streženju slik virtualnih naprav (ang. virtual machine image), na katerih je shranjen operacijski sistem in poljuben nabor programov. Na podlagi slike virtualne naprave lahko ustvarimo več duplikatov ali instanc.

Neutron je namenjen streženju virtualnih omrežij. Povezuje mrežne napravame, katere nudi Nova. V primeru bare-metal hypervisorja nudi predzagonsko okolje komponenti Ironic, ki skrbi za namestitev sistemskih slik direktno na strojno opremo (ang. bare metal provisioning).

Swift je objektno skladišče, ki sluzi shranjevanju varnostnih kopij vsebin bločnih naprav, podatkovnih baz in slik virtualnih naprav.

Medsebojno odvisnost opisanih komponent prikazuje slika 1.



Slika 1: OpenStack arhitektura.

2.4 Virtualizacija CUDA naprav

OpenStack Nova podpira različne hypervisorje za streženje virtualnih računskih naprav, od katerih je za poganjanje procesov, ki se izvajajo na grafičnih procesorjih, najprimernejši LXC ali Ironic.

LXC spada v skupino psevdo-virtualizacij, ker gostujoči operacijski sistem za delovanje uporablja jedro gostitelja. Posledično to pomeni, da ustvarjen virtualni računalnik ali v danem primeru kontejner (ang. container) ne omogoča poljubne izbire gostujočega operacijskega sistema. Nabor je omejen zgolj na Linux distribucije, ki imajo združljivo verzijo kernela. Druga negativna posledica je možnost kršenja kvot porabe strojnih sredstev. V zameno ponuja skoraj nično zmanjšanje performančnih zmogljivosti in hitro odzivnost oblačnega sistema.

OpenStack Nova, poleg LXC podpira tudi namestitev v naprej pripravljenih sistemskih slik direktno na strojno opremo (Ironic), kar je tudi naj-

primernejši način za HPC oblak in poganjanje CUDA 7 aplikacij. Na ta način, lahko hitro in ekonomično vklopimo želeno število fizičnih strežnikov, na njih namestimo želeni operacijski sistem in storitve. Aplikacije v takem okolju delujejo s polno hitrostjo (ang. native speed). Dodatna prednost tega pristopa je tudi v tem, da ni omejen pri izbiri operacijskih sistemov, podpira praktično vse operacijske sisteme, kompatibilne z izbrano strojno opremo in možnostjo namestitve preko mrežne povezave (PXE). Ironic ze delovanje potrebuje naslednje strojne komponente:

IPMI običajno je integriran le na strežniških računalnikih, služi oddaljeni kontroli in dostopu do strežnika. Omogoča vklop/izklop strežnika, oddaljen priklop na zaslonsko sliko, oddaljeno uporabo tipkovnice in miške in spreminjanju nastavitev strojne inicializacije in zagona (ang. BIOS).

PXE boot ali zagon računalnika preko mrežne povezave. Omogoča prenos in namestitev v naprej pripravljene sistemske slike, preko mrežne povezave v spomin strežnika, nadomesti branje sistema iz trdega diska, ali drugih fizično priključenih spominskih medijev.

najmanj dva mrežna vmesnika eden namenjen PXE zagonu in drugi za splošni dostop strežnika do omrežja.

izolirano mrežno infrastrukturo namenjeno le PXE zagonu sistemov na strežnikih.

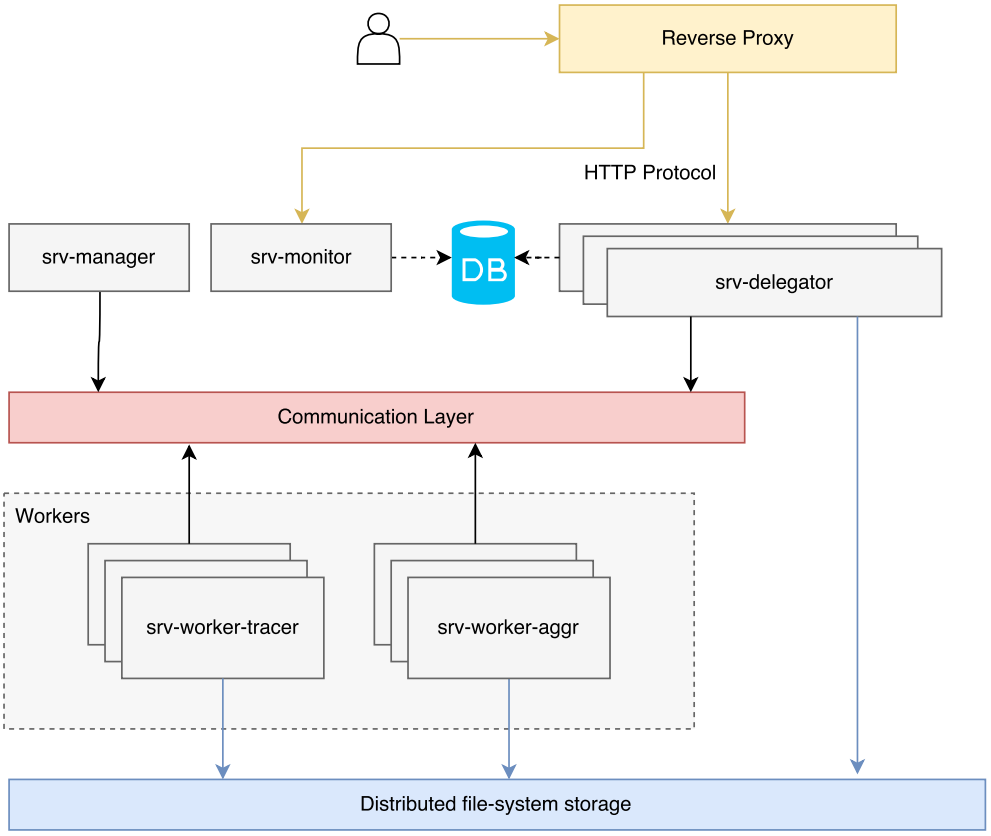
Alternativa LXC virtualizacije in Ironic-a je Xen, VMWare in vSphere, ki izrabljajo strojno virtualizacijo. Poleg slabših performančnih rezultatov, omenjene tehnologije niso povsem proste za uporabo v komercialne namene, zato jih v tem programskem sklopu projekta nismo obravnavali.

2.5 Izbira oblačne infrastrukture

Za poganjanje ART aplikacije bomo uporabili odprtokodno rešitev OpenStack s hypervisorjem Ironic in se tako kar najbolje približali HPC GPU računskemu sistemu. S tem namenom bomo nadgradili obstoječo infrastrukturo z dodatnimi strežniki, ki bodo vsebovali tudi grafične procesorje in IPMI sistem.

3 Arhitektura sistema

Zasnova sistema temelji na modularni in skalabilni arhitekturi za paralelno izvajanje CUDA operacij. Na sliki 2 so prikazane predvidene komponente in njihova soodvisnost. Diagram moramo razumeti kot abstraktni prikaz želenega delovanja sistema, na osnovi katerega bomo izbrali najprimernejše odportokodne rešitve. Po potrebi bomo razvili dodatne manjkajoče dele ali na novo izdelali komponente v primeru, da obstoječe ne bo dovolj prilagojeno za potrebe projekta.



Slika 2: Predvidena arhitektura sistema.

Različne algoritme distribuiramo na način, da jih predrugačimo in razdelimo na medsebojno neodvisne dele. Enosmerni ray-tracing algoritmi (ang. Forward Ray-Tracing), kateri je uporabljen v projektu ART, so že po naravi deljivi na poljubno število neodvisnih delov, pri katerem vsak predstavlja eno ali več poti izbranega radijskega žarka. Rezultat ART simulacije je receptorska ravnina, ki predstavlja dvodimenzionalen niz valovnih vektorjev in pripadajočih faz. Vsak posamezen žarek sledimo, dokler ne prekorači dolo-

čenega števila odbojev ali zadene receptorske ravnine, kjer se karakteristika žarka združi z obstoječim podatkom v receptorski ravnini.

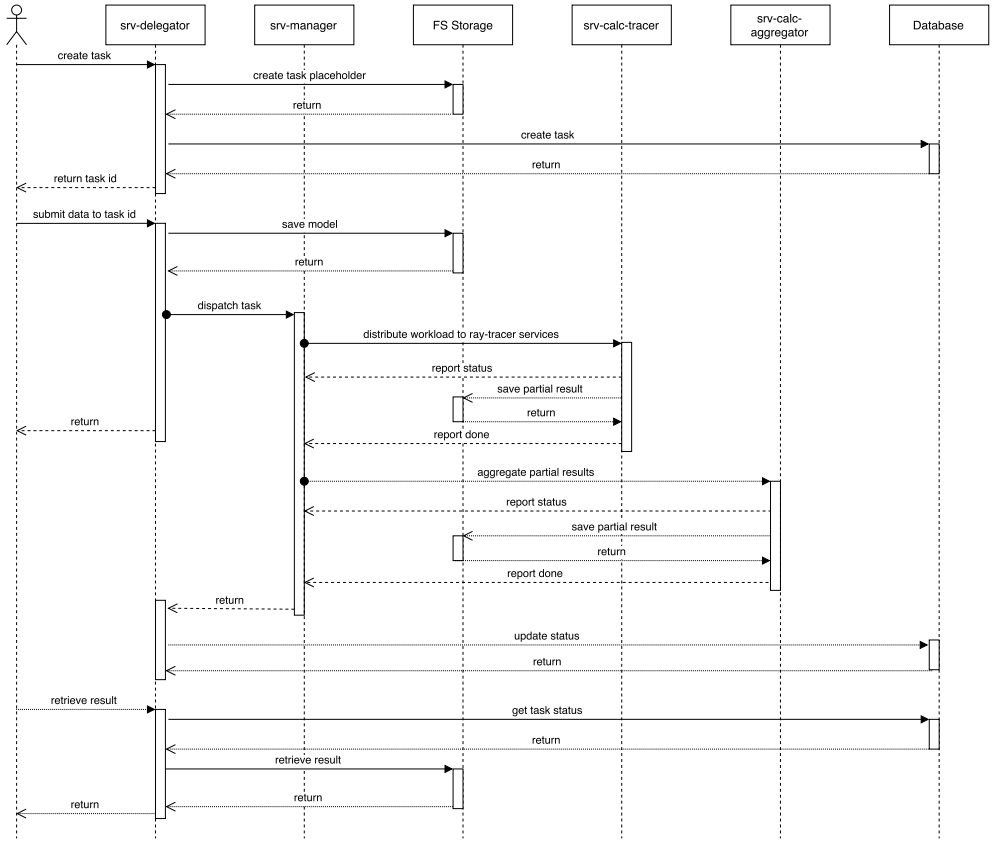
Za porazdeljeno izvajanje simulacije na več napravah je potrebno obstoječo logiko, napisano v CUDA, prenesti in ločiti na dva procesa, oziroma na dve komponenti (Tracer in Aggregator). Prvi proces generira žarke in izračunava karakteristike pri odboju, lomu, uklonu in transmitivnosti žarka skozi snov, drugi pa izračunava superpozicije vpadnih žarkov na receptorski ravnini. Prvi proces je že po naravi horizontalno skalabilen, ker operira na istem setu vhodnih podatkov (konfiguracija, geometrija), pri čemer se z vsakim novim procesom linearno poveča število obdelanih žarkov glede na čas. Drugi proces je možno porazdeliti glede na vhodne podatke po sektorjih receptorskih ravnin. Od posamezne Tracer in Aggregator komponente se pričakuje maksimalen izkoristek ene CUDA naprave. Tako lahko eksaktno določimo porabo strojnih sredstev in posledično optimalno razporeditev instanc komponent med napravami.

Za način distribuiranja nalog smo izbrali programski model map-reduce, pri katerem se osnovni problem razcepi na več vzporednih in medsebojno neodvisnih opravil 'map', ki se izvajajo na ločenih strojnih ali virtualiziranih sredstvih. Pred zaključkom naloge, se delni podatki agregirajo 'reduce' v končni rezultat.

Prvi korak v prikazanem modelu (slika 2) izvede Manager komponenta, ki je zadolžena za spremljanje tekočih nalog, opravil in komponent tipa Worker. Manager na podlagi razpoložljivih strojnih sredstev presodi, na koliko opravil razbiti nalogo, preveri razpoložljiva sredstva in njihovo obremenjenost, ter razporedi opravila na podlagi pridobljenih informacij. Po pridobljenih delnih rezultatih Manager preda delo komponenti tipa Aggregator, katera sestavi receptorsko ravnino. Podroben prikaz poteka je prikazan na sliki 3.

Za vse zahtevke s strani uporabnikov se pričakuje, da se posredujejo preko Reverse Proxy prehoda do instanc aplikacijskih strežnikov (Delegator instanc). Zaradi možnosti skaliranja (ang. service scaling), Reverse Proxy upravlja tudi nalogo razporejanja zahtevkov po načinu 'round robin'. Aplikacijski strežnik skrbi za kreiranje taskov v podatkovni bazi in shranjevanje vnosnih podatkov v datotečni sistem, iz katerega Worker komponente zajemajo vhodne podatke.

Komponente ki poganjajo algoritem so spisane v C/C++ programskem jeziku in uporabljajo CUDA + OptiX knjižnice. Ostale komponente bodo po potrebi narejene v JavaScript jeziku zaradi hitrejšega razvoja in nizkih strojnih zahtev. Za Reverse Proxy bo uporabljen NginX, ker nudi dobre



Slika 3: Predviden potek zahtevka simulacije.

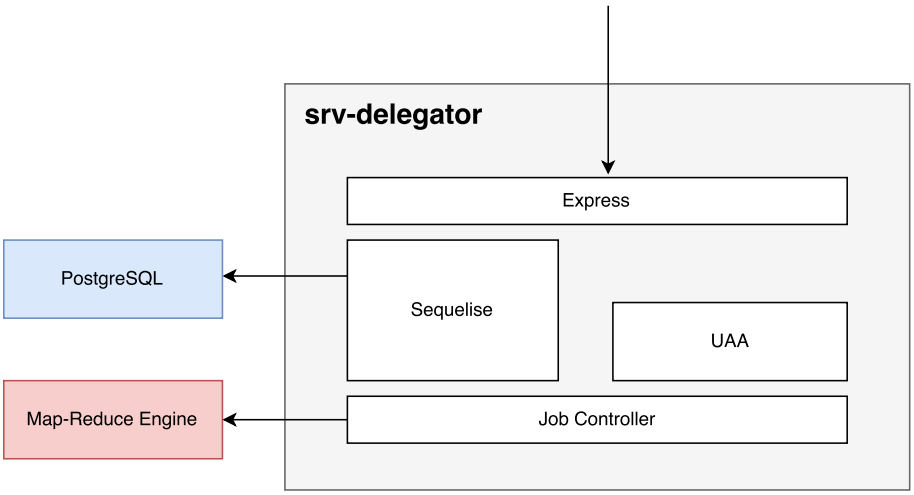
performančne zmogljivosti.

Preučili in predstavili smo dve možni implementaciji programskega modela map-reduce (Hadoop in Docker Swarm). Podroben opis arhitekture posamezne se nahaja v poglavjih 3.3 in 3.2, končna odločitev in argumentacija primarne izbire pa v poglavju 5.

3.1 Aplikacijski strežnik

Komponenta Delegator ima nalogo sprejemanja zahtevkov, shranjevanje podatkov v datotečni sistem in podatkovno bazo, ter nudi zunanji aplikacijski vmesnik za upravljanje in streženje rezultatov.

Po uspešno pridobljenih podatkih o geometrijskem modelu, radijskih izvorih in nastavitvah, delo delegira Apache Hadoop komponenti preko Job Controller modula.



Slika 4: Diagram Delegator komponente.

Komponenta definira zunanji aplikacijski vmesnik, preko katerega je možno upravljati simulator in oddajati delovne naloge preko HTTP REST protokola. Podatkovni format je definiran v JSON-u, razen geometrijskega modela. Definirane vstopne točke (ang. endpoints) REST API-ja so:

post /api/tracer/tasks Ustvari novi zahtevek za simulacijo. Vsebina POST zahtevka vsebuje JSON z informacijami o lokaciji, polarizaciji in jakosti radijskih sevanj ter zahtevani natančnosti in gostoti rezultatov receptorske ravnine.

get /api/tracer/tasks Če ima uporabnik administratorske pravice, metoda vrne vse naloge. Rezultat je niz identifikatorjev posameznih nalog.

get /api/tracer/tasks/:id Vrne trenutno stanje zahtevka v izvajanju ali statistiko izvajanja. Stanje zahtevka je lahko v čakanju, v izvajanju ali končan. Stanje izvajanja je opredeljeno glede na odstotek celotnega dela in kratkim opisom faze.

post /api/tracer/task/:id/data Služi za nalaganje geometrijskih modelov okolice. V telesu zahtevka se pričakuje datotečni format, ki opisuje geometrijo. Trenutno je to po meri narejen model v XML obliki, katerega bomo skušali spremeniti v standardiziran geometrijski format OBJ, PLY ali podobno.

get /api/users Vrne vse uporabnike v sistemu. Operacija je dovoljena samo administratorju.

post /api/users Na podlagi uporabniškega imena in gesla ustvari uporabnika. Operacija je dovoljena samo administratorju.

get /api/users/:id Na podlagi identifikacijske kode vrne uporabnika in pripadajoče meta-podatke.

post /api/users/auth/token Služi za avtentikacijo uporabnikov. Podprt bo OAuth2 način 'Resource Owner Password Credentials Grant'

Za komponento Delegator se predvideva uporaba programskega jezika JavaScript in ogrodja Express. Za komunikacijo z bazo in mapiranje objektno relacijskih struktur bomo uporabili Sequelise, task Controller modul bo deloval kot vmesnik do Hadoop instanc. Glede na vhodne parametre bo sestavil nalogo, shranil vhodne podatke v datotečni sistem HDFS in prožil izvajanje z uporabo Apache Hadoop knjižnic.

Pomembna naloga Delegator komponente je tudi avtentikacija in avtorizacija uporabnikov (UAA). Predvidena je uporaba OAuth2 standarda, ki točno definira zahtevke in načine avtorizacije. Predvidena sestava in odvisne komponente so prikazane na sliki 4.

3.2 Hadoop implementacija

Hadoop je odprtokodni sistem za porazdeljeno shranjevanje in procesiranje podatkov po programskem modelu map-reduce. Sestavljen je iz skupnega paketa, ki vsebuje OS podporo, map-reduce pogon (YARN), HDFS datotečni sistem. Skupni paket se nahaja na vseh instancah Hadoop gruče (ang. Hadoop Cluster). Instance v gruči se ločijo gleden na vloge in aktivne procese.

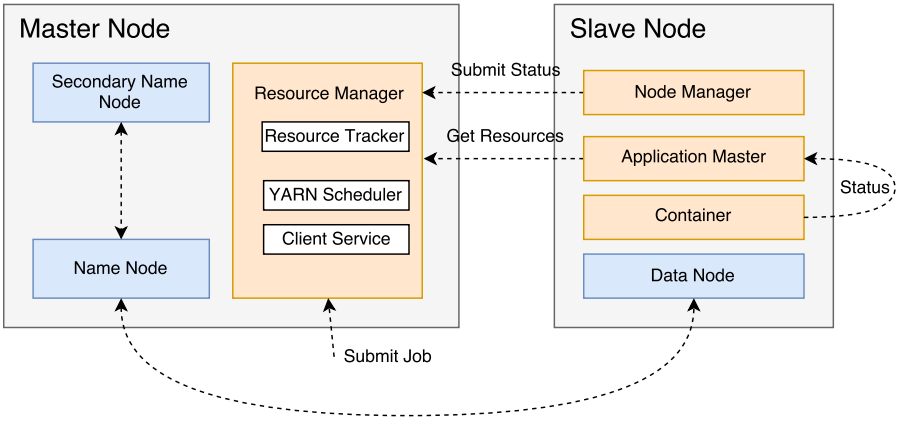
Osnovne komponente Hadoop orodja so:

- NameNode
- Secondary NameNode
- DataNode

- TaskTracker
- JobTracker
- ResourceManager
- NodeManager
- WebAppProxy
- MapReduceJobHistoryServer

Poznamo SlaveNode in MasterNode instance. Na vsaki SlaveNode instanci se izvajajo NodeManager, DataNode, TaskTracker procesi. Prvi je zadolžen za usklajevanje z ResourceManager-jem in posledično razporejanjem opravil. NameNode, Secondary NameNode in DataNode komponente skupaj koordinirajo HDFS distribuirani datotečni sistem. NameNode vsebuje in upravlja meta-podatke, kot so podatki o datotečnih blokih in imenski prostori (ang. namespece informations), DataNode pa skrbi za shranjevanje podatkov na posamezni SlaveNode instanci. Podrobnejši opis o delovanju datotečnega sistema se nahaja v poglavju 3.4.

Map-Reduce pogon ali poimensko YARN, uporablja en proces JobTracker-ja, preko katerega klient dodeljuje naloge Hadoop gruči. Ta komunicira s TaskTracker-ji, ki se nahajajo na posameznih SlaveNode instancah in izvršujejo opravila tipa 'map' in 'reduce'. Opravilo tipa 'map' se sovpada s komponento Tracer 'reduce' pa s komponento Aggregator (slika 2).

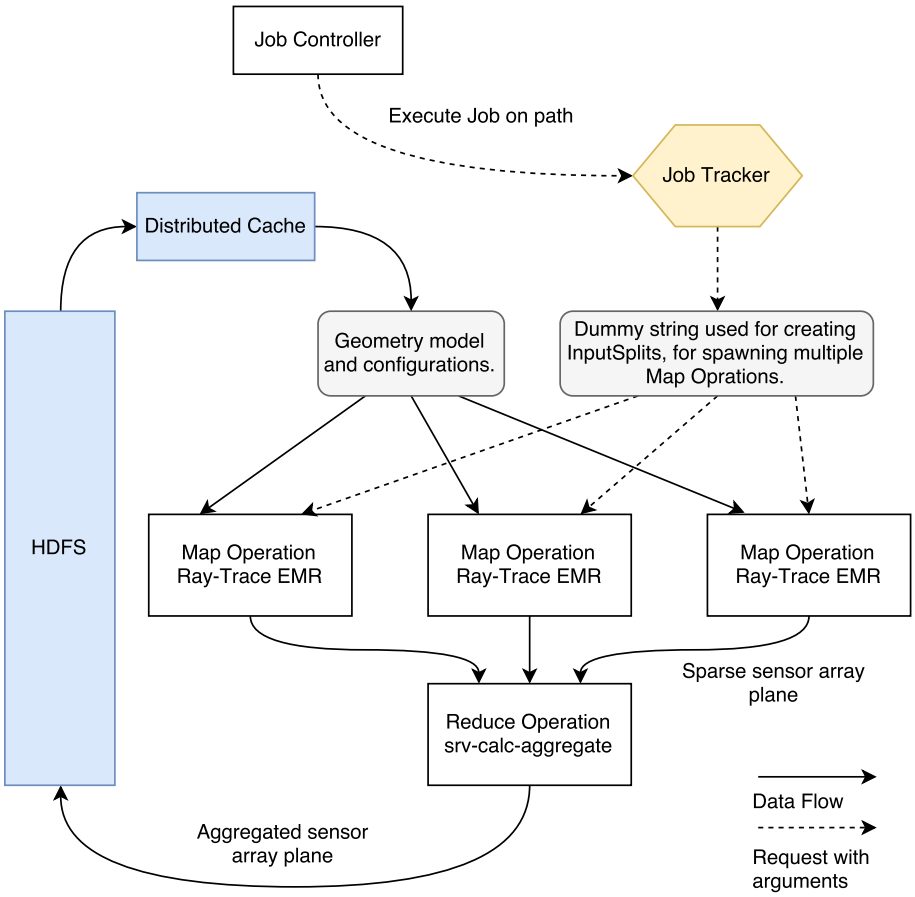


Slika 5: Prikaz komponent in soodvisnosti Hadoop storitev (HDFS, Map-Reduce).

Apache Hadoop je v osnovi zasnovan za obdelovanje in analizo velike količine podatkov (ang. Big Data). Naš primer je nekoliko drugačen, zaradi relativno majhne količine vhodnih podatkov in visoke računske zahtevnosti

na grafičnih procesorjih. Zaradi omenjenih razlik, bo potrebno prilagoditi Apache Hadoop map-reduce pogon (YARN).

Prva težava je, da Resource Manager trenutno ne vsebuje podatkov o grafičnih napravah, kar lahko ima za posledico slabo delovanje YarnScheduler-ja, ki je zadolžen za razporejanje opravil. Sistem zaradi tega lahko napačno predvideva prosta sredstva na Hadoop delovnih instancah (uporablja se tudi angleški izraz 'node'), ter tako zaradi napačno razporejenih opravil nekajkrat podaljša čas izvajanja naloge. Algoritem projekta ART se skoraj v celoti izvaja s CUDA kerneli, zato lahko CPU sredstva zanemarimo in namesto virtualnih jeder (ang. virtual cores vCore) navajamo razpoložljive grafične naprave. Tako prisilimo YarnScheduler da razporeja opravila glede na GPU naprave.



Slika 6: Izvajanje ART aplikacije v Yarn map-reduce pogonu.

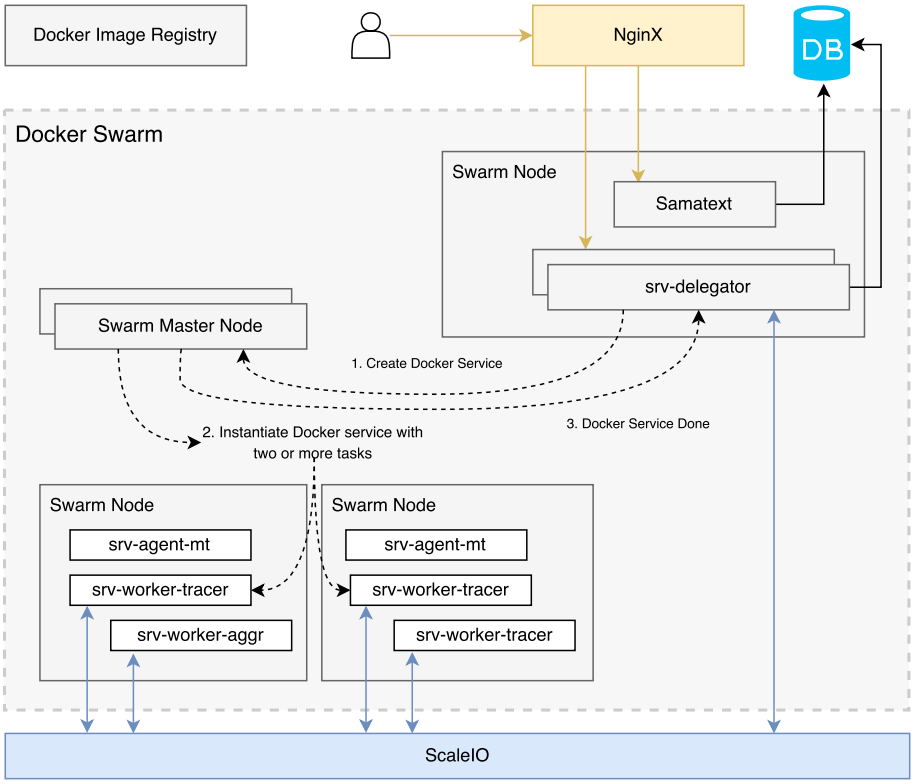
Druga težava oziroma značilnost je, da Apache Hadoop sestavi opravila tipa

'map' glede na vhodne podatke. Ob zagonu aplikacije se kot vhodni podatek navede niz poti do datotek ali map, na podlagi katerih se naredijo InputSlice objekti. Vsak predstavlja del vhodnih podatkov, največkrat posamezno vrstico v datoteki ali pa celotno datoteko, če je vhodni podatek pot do mape. InputSlice objekti se injektivno preslikajo v opravila tipa 'map', katera YarnScheduler distribuira med naprave. V našem primeru je vhodni podatek geometrijski model, katerega brez spreminjanja algoritma, ne moremo razdeliti na medsebojno neodvisne dele za procesiranje. Teoretično bi lahko prostor razbili na posamezne sektorje med katerimi bi med vsako iteracijo prenašali žarke. Na ta način bi omogočili izvajanje algoritma nad geometrijo, ki po velikosti presega pomnilniško kapaciteto posamičnega računalnika, vendar bi se zaradi prenašanja velike količine podatkov med iteracijami znatno upočasnilo celotno izvajanje. Ločitev vhodnih podatkov na posamezne dele pride v poštev samo takrat, kadar zaradi strojnih omejitev ne gre drugače.

3.3 Swarm implementacija

Kot alternativa Hadoop sistemu se ponuja sodobnejša rešitev, ki izvira iz sveta mikro storitev in temelji na sistemski psevdo-virtualizaciji operacijskih sistemov. Najbolj znana razširitev omenjene tehnologije je Docker, ki v osnovi definira izgradnjo posnetka kontejnerja (Docker Container Image). S časom se je tehnologija razširila na področje virtualizacije omrežij in orkestracije kontejnerjev. Sprva se je ta funkcionalnost ponujala kot ločena rešitev, imenovana Docker Swarm, ki so jo kasneje integrirali v Docker in preimenovali v Docker Swarm mode.

Docker Swarm omogoča povezovanje Docker pogonov v gručo, katero nadziramo na podoben način kot lokalno namestitev. Omogoča nam definirati Docker storitve (ang. Docker Service), ki ponazarjajo skupino Docker kontejnerjev, virtualna omrežja, katerim omenjene storitve pripadajo in storitve za razresevanje domenskih imen. Tako vsaka Docker Storitev dobi svoj naslov preko katerega se zahtevki porazdelijo med pripadajoče kontejnerje. Razporejanje obremenitev lahko poteka preko resolucij domen ali preko VIP. Docker Swarm Scheduler razporeja kontejnerje po instancah glede razpoložljiva sredstva, kar zagotavlja uravnoteženo obremenjeno gručo instanc pri frekventnem ustavljanju in poganjanju kontejnerjev.



Slika 7: Prikaz Docker Swarm arhitekture map-reduce modela.

S pomočjo Docker tehnologije lahko vsako Worker komponento zapakiramo v Docker Image, katero navedemo skupaj z argumenti ob kreiranju Docker Service-a. Ta samodejno glede na podane argumente zažene želeno število kontejnerjev, ki preko datotečnega sistema zajemajo vhodne podatke in shranjujejo rezultate, kot je to prikazano na sliki 3 in 2.

Sematext orodje se lahko uporabi za spremljanje porabe in beleženje zgodovine strojnih sredstev in tako nudi implementacijo Monitor komponente.

3.4 Datotečni sistem

V datotečni sistem se predvideno shranjujejo vsi vhodni, delni in končni rezultati simulacije. Poleg tega služi tudi kot arhiv preteklih rezultatov. Glede na izbrano implementacijo map-reduce programskega modela, smo se odločili za HDFS in ScaleIO.

Hadoop nudi distribuirani datotečni sistem HDFS, katerega sestavljajo DataNode, NameServer in Secondary NameServer komponente. DataNode storitev se izvaja na SlaveNode instancah in skrbi za branje in pisanje podatkov

na bločno napravo. NameNode vodi evidenco hranjenih datotek, število replikacij in ostale meta-podatke, ki so ključnega pomena za delovanje gruče. SecondaryNameNode služi kot podpora NameNode-u za shranjevanje preteklih operacij na datotečnem sistemu. Vse storitve MasterNode instance HDFS sistema se lahko poganjajo na večih napravah hkrati za zagotavljanje neprekinjenega delovanja (ang. High-availability).

Kot najbolj primeren distribuiran datotečni sistem pri uporabi Docker Swarm map-reduce arhitekture, smo izbrali ScaleIO, ker v primerjavi z alternativami ponuja visoko število IOPs (ang. Input Output Operations) in nizke latence. Specializiran je samo za virtualizacijo bločnih naprav in omogoča adekvatno količino sočasnih povezav glede na število naprav. Velikost posameznega modela ali delnega rezultata v našem primeru ne bo presegala velikosti pomnilnika posamezne instance, kar daje v tem segmentu prednost Docker Swarm in ScaleIO arhitekturi, v primerjavi s HDFS sistemom, ki je prilagojen za shranjevanje večjih datotek.

3.5 Optimizacije

Ray-tracing algoritem projekta ART združuje posamezne zarke v valovne fronte, kar pohitri izračun ali izboljša kvaliteto rezultata. Da to možnost izkoristimo v polni meri, je smiselno pri obeh izvedbah map-reduce modela žarke na posameznem 'map' opravilu zgostiti. To lahko dosezemo tako, da vsaka 'map' operacija generira samo žarke, ki pripadajo izbranemu kotnemu izseku. Te se določijo glede na število vzporednih opravil in željeni obseg izseva izvora radijskega valovanja.

Optimizacije so mozne tudi pri prenašanju in nalaganju geometrij. Smiselno bi bilo pretvoriti vhodni tekstovni datotecni format v binarni pred začetkom map-reduce opracije sledenja žarkom. Na ta način se rešimo večkratnega nepotrebnega branja in prevajanja modelov, zmanjšamo velikost vhodnih podatkov in tako razbremenimo opravila tipa 'map'.

4 Nameščanje oblačnih storitev

Oblačne storitve so skoraj vedno nameščene ali pa se izvajajo na gruči neodvisnih naprav z operacijskim sistemom. Za nameščanje, konfiguriranje in zaganjanje aplikacij in podpornih sistemov, posledično to pomeni zamudno večkratno izvajanje podobnih operacij v ukazni lupini, kar lahko privede do napak in nekonsistentnosti. Zaradi teh posledic in hitrejšega dela, poznamo orodja, katerim s pomočjo programskega jezika definiramo želeno

persistenčno stanje sistema.

4.1 Orodje Ansible

Eno izmed orodji na nameščanje, po našem mnenju primerno za potrebe projekta ART, je Ansible. Ansible uporablja YAML datotečni format za opis definicije sistema in SSH (ang. Secure Shell) za oddaljen dostop in uveljavljanje stanja. Primerno je tako za nameščanje Hadoop in Docker Swarm storitev. Pri implementaciji prvega primera se uporabi za namestitev Hadoop gruče, Delegator komponente in Cloudera orodja za spremljanje gruče. V drugem primeru Ansible namesti in na zahtevo horizontalno skalira Docker Swarm, ki za razliko od Hadoop-a vsebuje tudi Delegator in Sematext orodje, dodatno pa je potrebno namestiti Docker Registry in ScaleIO datotečni sistem.

V obeh primerih bomo uporabili PostgreSQL podatkovno bazo in implementirali Ansible skripte za namestitev.

4.2 Izgradnja Docker posnetkov in register

V tem poglavju so predstavljene poglavitne razlike med nameščanjem Hadoop in Docker Swarm map-reduce izvedbe.

Pri uporabi Docker Swarm izvedbi, so Worker komponente shranjene v obliki Docker Container posnetkov, za hranjenje katerih se uporablja Docker Registry. Ta mora biti dostopen iz vseh instanc Swarm gruče, da lahko Docker Engine pridobi sliko na podlagi katere ustvari kontejner. V registru se hranijo slike komponent tipa Worker in tudi ostale podporne komponente. Izvzet je samo Reverse Proxy in podatkovna baza. Namesto namescanja z uporabo Ansible orodja se uporabi Dockerfile datotečni zapis, ki vsebuje potrebne informacije za igradnjo slik. Na ta način lahko hitro horizontalno skaliramo tudi podporne komponente sistema.

5 Sklep

V začetku tega poglavja smo začrtali programsko arhitekturo in kasneje predstavili dve, po našem mnenju trenutno najbolj primerni implementaciji za izvajane ray-tracing algoritmov na grafičnih procesorjih. V tem poglavju bomo predstavili prednosti in slabosti vsake, ter argumentirali končno odločitev.

Docker Swarm je relativno nov upravitelj gruče kontejnerjev. Uporablja Docker, ki v določenih pogledih še ni najbolj zanesljivo orodje za upravljanje

s kontejnerji, in se po večini ne uporablja v produkcijskih okoljih. Kljub zelo preprosti namestitvi, Docker Swarm nudi širok nabor funkcionalnosti. Najbolj poglavitne so razporejanje kontejnerjev, izdelava virtualnih omrežij, resolucijo domenskih imen in razporejanje obremenitev. Posamezne storitve zaradi orodja Docker tečejo v izoliranih okoljih, ki zmanjšujejo možnost sesutja posamezne instance zaradi napake ene operacije, oziroma instance komponente. Za delovanje ne zasede veliko strojnih sredstev in se kljub velikemu številu virtualnih naprav s povprečnimi ali podpovprečnimi strojnimi specifikacijami odziva dokaj hitro.

Ker gre za dokaj novo in na trenutke še nedovršeno tehnologijo, ne nudi dobrih odprtokodnih ali prostih orodji za pregled in nadzor dogajanja v gruči. Dodatno bi bilo treba nastaviti ScaleIO bločni sistem in ga uporabljati v Docker instancah, kar lahko privede do zapletov. Isto velja za CUDA knjižnice in dostop do grafičnih procesorjev.

Hadoop je po drugi strani dokaj zrelo in namensko orodje za izvajanje map-reduce programskega modela. Sestavlja ga množica storitev, ki olajša delo končnemu uporabniku, otežuje pa namestitev in vzdrževanje sistema. Posamezne storitve je potrebno ročno prilagoditi za zagotavljanje redundance. Primarni jezik opravil Hadoop je Java, vendar obstajajo tudi C++ knjižnice. Sistem brez obremenitev porablja občutno več strojnih sredstev kot Docker Swarm, kljub primerljivim odzivnim časom zaganjanja preprostih nalog. Dobro se povezuje z datotečnim sistemom, kar omogoča pametno razporejanje instanc komponent glede na vhodne podatke. Te prednosti v našem primeru ne moremo izrabiti, ker instance Tracer operirajo na istem naboru vhodnih podatkov, vendar pa pušča odprto možnost za izvajanje simulacij na obsežnih podatkih. V tem primeru bi bilo potrebno predrugačiti in dopolniti ray-tracing algoritem.

Zaradi namenske specializiranosti za programski model map-reduce, odprtih možnosti za razširitev simulacije na obsežne vhodne geometrije in večje stabilnosti sistema, smo se odločili za Hadoop implementacijo.

6 Literatura

Alam A in Ahmed J (2014). Hadoop architecture and its issues. 2014 international conference on computational science and computational intelligence, 2: 288-91. http://ieeexplore.ieee.org.nukweb.nuk.uni-lj.si/document/6822351/. <18.02.2017>

Bias R (2015). Cloudscaling blog. Killing the storage unicorn. http://cloudscaling.com/blog/cloud-computing/killing-the-storage-unicorn-purpose-built-scaleio-spanks-multi-purpose-ceph-on-performance/. <18.02.2017>

Coppa E (2017). Hadoop internals. Anatomy of a mapreduce job. http://ercoppa.github.io/HadoopInternals/AnatomyMapReduceJob.html. <18.02.2017>

Mathew J, (2016). Spark my cloud. A comparative review of Swift vs Ceph. http://www.sparkmycloud.com/blog/a-performance-review-of-swift-vs-ceph/. <18.02.2017>

Mu Q, Jia Y in Luo B (2015). The optimization scheme research of small files storage based on HDFS. 2015 8th international symposium on computational intelligence and design (ISCID). http://ieeexplore.ieee.org.nukweb.nuk.uni-lj.si/document/7468985/. <18.02.2017>

Northam L, Smits R, Daudjee K in Istead J (2013). Ray tracing in the cloud using mapreduce. 2013 international conference on high performance computing & simulation (HPCS): 19-26. http://ieeexplore.ieee.org.nukweb.nuk.uni-lj.si/document/6641388/. <18.02.2017>

OpenStack. OpenStack Fundation (2017). Introduction to Ironic. https://docs.openstack.org/developer/ironic/deploy/user-guide.html. <18.02.2017>

Shi L, Chen H in Sun J (2009). vCUDA: GPU accelerated high performance computing in virtual machines. 2009 IEEE international symposium on parallel & distributed processing, Papers 8: Patents: 1-11. http://ieeexplore.ieee.org.nukweb.nuk.uni-lj.si/document/5161020/. <18.02.2017>

Shirahata K, Sato H in Matsuoka S (2010). Hybrid map task scheduling on GPU-based heterogeneous clusters. 2010 IEEE second international confe-

rence on cloud computing technology and science, Papers 14: 733-40. http://ieeexplore.ieee.org.nukweb.nuk.uni-lj.si/document/5708524/. <18.02.2017>

Reza M, Sinha A, Nag R in Mohanty P (2015). CUDA-enabled Hadoop cluster for sparse matrix vector multiplication. 2015 IEEE 2nd international conference on recent trends in information systems (ReTIS): 169-72. http://ieeexplore.ieee.org.nukweb.nuk.uni-lj.si/document/7232872/. <17.02.2017>