DOKUMENTACIJA ART SISTEMA, NAVODILA ZA UPORABO

Tehnično poročilo

Revizija:	1.2
Projekt:	ARRS-ART
Avtorji:	Vito Čuček, Marjan Šterk, Matic Herman, Gregor Pipan
Datum:	11. april 2018

Kazalo

1 Uvod **1**

2 Infrastruktura **2**
2.1 Infrastruktura kot storitev 2
2.2 Uporabljena testna infrastruktura sistema 2

3 Arhitektura sistema **4**
3.1 Aplikacijski strežnik 7
3.2 Koordiniranje nalog 10
3.3 Podatkovna baza 12
3.4 Podatkovni posrednik 12
3.5 Datotečni sistem 14
3.6 Porazdeljeno izvajanje algoritma 15
3.7 Agregacija delnih rezultatov 16

4 Uporabniška navodila **17**
4.1 Namestitev sistema 17
4.2 Priprava geometrijskega modela 18
4.3 Priprava vhodnih podatkov in izvedba simulacije 19

5 Literatura **24**

Kazalo slik

1 Arhitektura sistema 5
2 Potek zahtevka simulacije 6
3 Diagram aplikacijskega streznika. 7
4 Diagram komponente Manager 10
5 Shema poteka komunikacije od začteka do konca izvajanja posla 11
6 Shema podatkovne baze managerja (srv-manager). 12
7 Shema podatkovne baze aplikacijskega strežnika (srv-delegator). 13
8 Izvoz geometrijskega modela 19
9 Prikaz izgub radijskega signala 20
10 Spletni vmesnik in dokumentacija API-ja 23
11 Porženje zahtevkov preko spletnega vmesnika 23

Slovar izrazov

Izraz	Pomen
Hypervisor	Program za upravljanje in izgradnjo virtualnih strojev.
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 2.2 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 razvitega sistema in projekta je karakterizirati radijsko okolje in izboljšati radijsko lokalizacijo s pomočjo naprednih tehnik sledenja žarkom.

V naslednjih poglavjih bomo opisali arhitekturo sistema, obrazložili ključne sestavne dele in opisali izbrano strojno infrastrukturo. V prejšnjem programskem sklopu projekta (2.1) smo nekaj omenjenih rešitev in idej za izvedbo že podali, tokrat pa se osredotočamo na konkretno izvedbo rešiteve, ki služi kot testno okolje za poganjanje simulacij sledenja žarkom (ang. Ray-Tracing).

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

V okviru programskega sklopa 2.2 smo pripravili testno infrastrukturo v oblaku, razvili in testirali vse potrebne programske komponente, implementirali mehanizem za prevajanje in namestitev sistema ter izboljšali in razčlenili delovanje algoritmov za potrebe vzporednega izvajanja. Slednje se predvsem nanaša na separacijo procedur na posamezne neodvisne komponente in zagotavljanje večje stabilnosti večnitnega izvajanja. Dopolnili smo visokonivojski načrt sistema in na podlagi implementirane rešitve dodali podrobne razlage posameznih komponent in uporabljenih protokolov.

Za potrebe hitrejšega in celovitega testiranja smo izdelali vtičnik odprtokodenga programa Blender, ki poenostavlja izdelavo vhodnih modelov simulacije ali pretvorbo in parameterizacijo že obstoječih geometrij, zapisanih v OBJ, 3DS, FBX, PLY, formatih Collada. Namestitev in uporaba vtičnika se nahaja v poglavju 4.2 (Priprava geometrijskega modela).

2 Infrastruktura

V okviru aktivnosti 2.1 smo analizirali in opisali odprtokodne rešitve za vzpostavitev obsežnega okolja kot platforme za poganjanje algoritmov na graficnih cevovodih. V tem poglavju bomo na kratko obnovili ključne pojme oblačnih komponent in storitev, v poglavju 2.2 pa opisali testno okolje implementirane oblačne storitve sledenja žarkom.

2.1 Infrastruktura kot storitev

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.

Hitrost izračuna propagacije radijskih valov skozi prostor in poustvarjanje raznih pojavov je odvisno od strojne infrastrukture, programske implementacije in zahtevane natančnosti. Oblačna infrastruktura z ustrezno 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.

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.

2.2 Uporabljena testna infrastruktura sistema

Na projektu ART smo se odločili za sistem IaaS storitev VMware zaradi dobre podpore, fleksibilnosti dodeljevanja grafičnih virov in preprostega načina uporabe. Trenutno na tržišču obstaja več rešitev dodeljevanja grafičnih zmogljivosti virtualnim strojem, od katerih je večina omejena na posamezno

implementacijo hipervisorja, modela grafičnega procesorja in proizvajalca. VMware-ova platforma vCould nudi dokaj širok izbor tehnik in se na ta način po uporabnosti približa vodilnim ponudnikom spletnih IaaS storitev. S stališča ART sistema to omogoča, da se testira različne načine virtualizacije in zmogljivosti glede na utilizacijo strojnih virov.

Za testno okolje projekta ART smo uporabili GPU pass-through, ki posameznemu virtualnemu stroju preda vodilo PCI, preko katerega krmili grafični procesor in pripadajočo periferijo. Upravljanje je v celoti prepuščeno klientu hipervisorja, kar ima za posledico nezmožnost deljenja moči posamezne graficne enote.

Vsak fizični strežnik, ki poganja hipervisor vSphere, mora imeti zadostno število grafičnih kartic, neposredno dostopnih na vodilih PCI, ker številčnost GPU-jev določa zgornjo mejo števila virtualnih strojev na posamezen fizični strežnik. Posledično je potrebno za optimalno delovanje izrabiti vse razpoložljive vire na vsaki vzpostavljeni instanci, kar v praksi ne predstavlja večjega problema. V nasprotju z virtualizacijo API, pass-through nudi enak nabor funkcionalnosti in enako zmogljivost kot bare-metal sistemi.

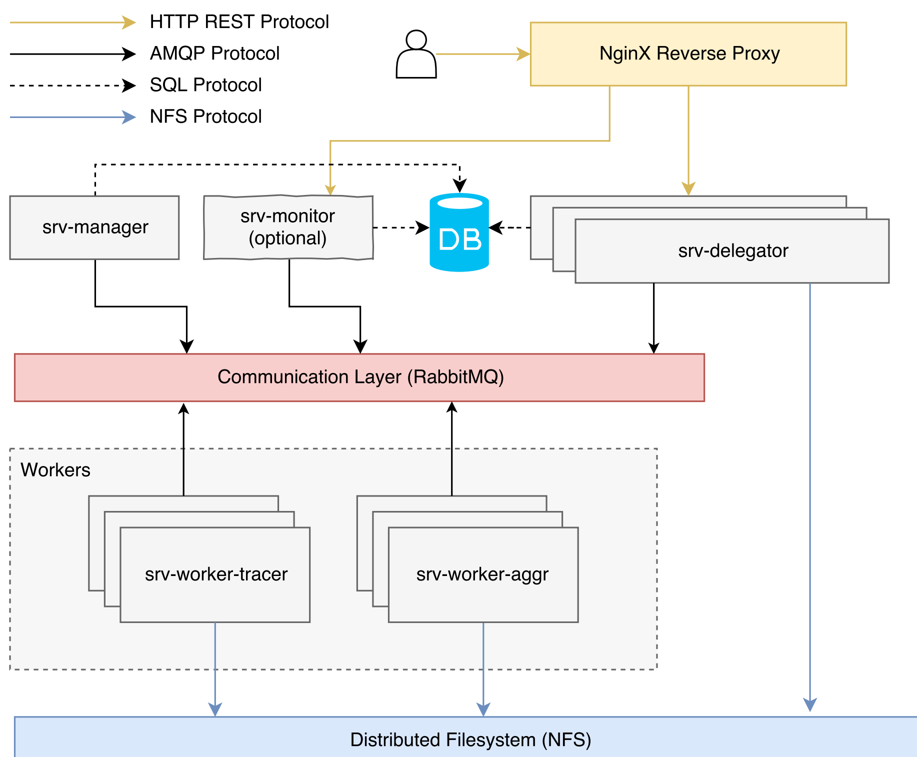
Uporabljeno testno okolje nam nudi širok nabor strojnih sredstev za testiranje skalabilnosti implementirane rešitve sistema ART, kar omogoča eksaktno preverjanje predpostavk prvotnih idej, ter medsebojne odvisnosti in komunikacije komponent. Zaradi zmožnosti hitre oddaljene manipulacije strojnih sredstev smo namenili posebno pozornost testiranju sistema ob izpadu posameznih komponent in na podlagi ugotovitev prilagodili komunikacijski protokol ter hrambo ključnih parametrov stanja sistema.

3 Arhitektura sistema

Zasnova sistema temelji na modularni in skalabilni arhitekturi mikro storitev (ang. microservices) za paralelno izvajanje. Kot smo že ugotovili v programskem sklopu 2.1, je za distribuirano izvajanje algoritmov sledenja žarkom najprimernejsi programski model 'MapReduce'. Sestavljata ga dve sekvenčni operaciji, pri čemer prva dani problem razdeli na poljubno število medsebojno neodvisnih nalog, druga pa skrbi za združitev delnih izračunov, pridobljenih pri prvi operaciji, in izgradnjo končnega rezultata. Te funkcije opravljajo v našem sistemu komponente tipa sledilnik (ang. tracer) in agregator, katerih delovanje nadzoruje in koordinira komponenta manager, kot je prikazano na sliki 2. Instance komponent, ki neposredno izvajajo algoritem sledenja žarkom ali sodelujejo pri obdelavi rezultata imenujemo delavci (ang. worker). V našem primeru se njihovi algoritmi izključno izvajajo na grafičnih procesorjih, ki lahko pripadajo enemu ali večim virtualnim strojem. Vsak proces tipa delavec (ang. worker) je implementiran na način, da skuša kar se da izrabiti grafične zmogljivosti enega procesorja, kar je glavno vodilo pri razporejanju nalog v algoritmu komponente manager. Posledično to pomeni, da programska horizontalna skalabilnost posamezne simulacije linearno sovpada s številom grafičnih procesorjev, oziroma z elastičnostjo infrastrukture do zgornje meje, ki je odvisna od kompleksnosti simuliranega modela (porabe grafičnega spomina). Da lahko podrobneje razumemo zgornjo trditev, je potrebno razložiti delitev prvotnega problema na manjše naloge in ustroj algoritma sledenja žarkom.

Primarno se uporabljene knjižnice sledenja žarkom (OptiX) izrabljajo za sledenje vidne svetlobe, ki ima drugačne karakteristike od radijskih valov. Pri vidni svetlobi je izrazit efekt med snovjo in svetlobo sipanje, ki v primeru simulacije pri vsaki interakciji potencira število žarkov v vse smeri in na ta način relativno izotropno zapolni prostor. Posamezen žarek v takšni simulaciji si lahko predstavljamo kot skupek fotonov, ki potuje skozi prostor in se na koncu akumulira na receptorsko ravnino.

Pri radijskem valovanju je efekt sipanja v naših okoliščinah redek pojav in prevladuje odboj, lom in uklon. Tem prevladujočim lastnostim sledi tudi integriran algoritem ART, ki za razliko od posameznih fotonov ali skupkov simulira valovne fronte. Ko valovna fronta doseže receptorsko sfero, ta vsebuje celotno informacijo o signalu na karakteristični poti, ki jo določa zaporedje interakcij z okolico. Efekti loma, uklona in odboja pripadajo ločenim valovnim frontam tako kot tudi različna zaporedja interakcij z ravninami, ki definirajo obliko in material okolice simuliranega modela. Bistvena razlika, ki vpliva na porazdeljeno izvajanje algoritma, nastopi pri akumulaciji signala na receptorski ravnini. Posamezna valovna fronta mora biti v receptorski



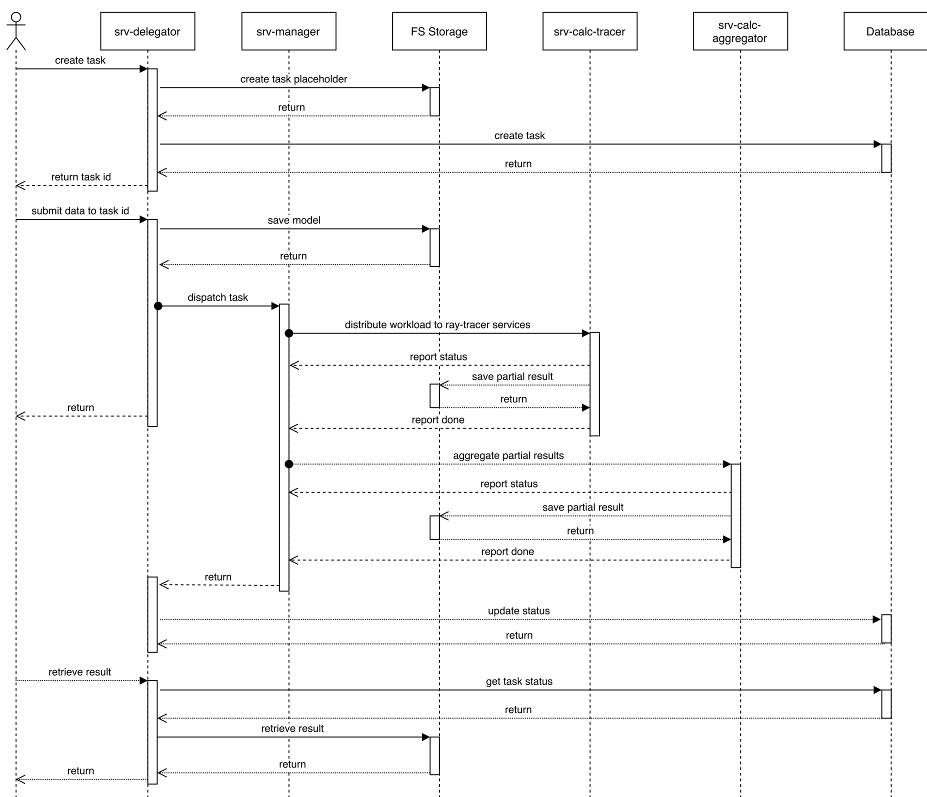
Slika 1: Arhitektura sistema

sferi upoštevana samo enkrat, ker že vsebuje intenziteto signala na karakteristični poti, medtem ko se valovne fronte različnih karakteristik seštevajo. To pomeni, da je potrebno pred zapisom prispevka v receptorsko sfero preveriti, ali je dana valovna fronta že upoštevana. V integriranem algoritmu zgornji problem rešuje bloom filter, ki hevristično z dokaj dobro natančnostjo rešuje težave s pomnilnikom, ne zaobide pa sekvenčnega procesa filtriranja valovnih front.

Pri paralelizaciji algoritma na posamezne virtualne stroje si zaradi latence in prepustnosti ne moremo privoščiti globalnega 'bloom filter' pomnilnika, tako kot si tudi zaradi prevelike količine podatkov ne moremo privoščiti izvedbe filtra na eni instanci. Rešitev je v predrugačenju algoritma na nacin, da na vseh vzporednih procesih sproducira disjunktne resitve, katere lahko kasneje brez filtriranja preprosto seštejemo. Konkretno to pomeni, da vsaka instanca sledilnika pri prvi interakciji upošteva samo določene ravnine modela. Po prvi interakciji z materialom in eventuelni izločitvi valovne fronte v dani smeri, algoritem deluje brez sprememb ekvivalentno na vseh procesih sledilnika. Na posamezni istanci je bloom filter še vedno potreben.

Manager, delegator in delavci se povezujejo na komunikacijski sloj, preko

katerega si izmenjujejo informacije. RabbitMQ je v dani situaciji najprimernejša izbira zaradi širokega nabora funkcionalnosti, izbire programskih jezikov, preverjenega delovanja in stabilnosti. Omogoča izvajanje oddaljenenih klicev procedur, kot tudi sporočanje statusov na tako imenovane 'exchange' filtre in 'queue' liste. Na ta način se povečuje fleksibilnost in razširljivost sistema, ker je možno brez večjih posegov naknadno integrirati komponente, ki poslušajo dogajanje v sistemu (skupnem vodilu) in nanj reagirajo.



Slika 2: Potek zahtevka simulacije

Za izmenjavo večjih količin podatkov smo uporabili distribuirani mrežni datotečni sistem NFS (Network File System), na katerega se shranjujejo delni in končni rezultati simulacije ter vhodni podatki o geometriji okolice, konfiguracija simulacije in lastnosti materialov.

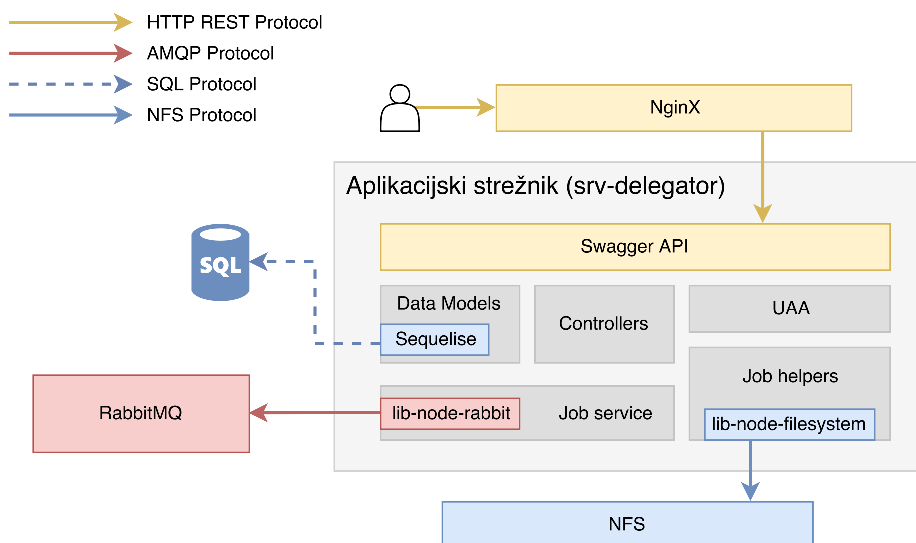
Delegator ima vlogo aplikacijskega strežnika in služi kot vstopna točka v sistem preko REST aplikacijskega vmesnika. Zaradi možnega večjega števila klientov je zasnovan kot skalabilna komponenta, ki se povezuje na komunikacijski sloj, datotečni sistem in podatkovno bazo. Skrbi za prijavo in avtentikacijo uporabnikov, streženje končnih rezultatov ter pošiljanje in pridobivanje statusov o zahtevkih. Za porazdelitev zahtevkov med instancami

delegatorja in morebitno preusmeritev na dodatne komponente smo uporabili povratni posredniški strežnik NginX.

3.1 Aplikacijski strežnik

Aplikacijski strežnik, poimenovan Delegator, ima nalogo sprejemanja zahtevkov, avtorizacijo in avtentikacijo uporabnikov, shranjevanje vhodnih podatkov v datotečni sistem in podatkovno bazo ter nudi zunanji aplikacijski vmesnik za upravljanje in streženje rezultatov. Implementiran je v programskem jeziku JavaScript z uporabo orodja NodeJS za razvoj strežniških aplikacij.

Aplikacijski vmesnik je generiran s pomocjo orodja Swagger in nudi možnost abstraktnega opisa končnih vstopnih točk (ang. end-points) in vhodnih modelov, ki skupaj definirajo aplikacijski protokol. Abstrakten opis protokola omogoča enostaven prenos v dugo razvojno okolje in hitro spremembo podatkovnega formata (JSON, XML, ...). Poleg široke fleksibilnosti uporabniku nudi spletno stran z dokumentacijo aplikacijskega vmesnika in orodje za testiranje delovanja zahtevkov.



Slika 3: Diagram aplikacijskega streznika.

Strežnik komunicira s podatkovno bazo PostgreSQL z uporabo ORM (ang. Object Relational Mapping) orodja Sequelise, ki nudi podporo migracij podatkovnih shem. Uporablja se za hranjenje in preverjanje dostopa uporabnikov do sistema in hrambo statusov poslanih uporabniških zahtevkov za simulacijo. Postopki preverjanja dostopa se nahajajo v modulu UAA (ang. User Authentication and Authorization) in zadostujejo standardu OAuth2.

Po uspešni avtorizaciji uporabnik pošlje zahtevek za simulacijo, katerega prevzame in validira orodje Swagger. Če vsebina ustreza definiciji protokola, strežnik dodeli zahtevku generično identifikacijsko kodo in ustvari nov vnos v podatkovni bazi, na katero se klient sklicuje pri pošiljanju podatkov o geometrijskem modelu okolice in materialih. Slednji podatki se posredujejo preko modula za upravljanje z datotekami (ang. Static file handlers) na distribuiran datotečni sistem, od koder so dostopni sledilnikom. Po zapisu parametrizirane geometrije, informacij o materialih in konfiguraciji simulacije, strežnik z uporabo knjižnice 'lib-node-rabbit' proži zahtevek za zagon. Delovanje knjižnice in nadaljnjih procedur za porazdeljeno izvajanje je opisano v poglavjih 3.2 in 3.4.

Za izmenjavo podatkov med strežnikom in klienti se uporablja JSON format z izjemo parametriziranega geometrijskega modela in rezultatov, ki uporabljajo zapis XML. XML zapis modela in materialov smo ohranili zaradi tesne povezanosti s samim algoritmom sledenja žarkom. V sklopu projekta smo izdelali urejevalnik, kateri omogoča pretvorbo in parametrizacijo geometrije iz standardnih geometrijskih formatov (OBJ, FBX, 3DS, ...) v XML, katerega podpira simulacija sledenja radijskih valov. Z isto komponento smo podprli prikazovanje rezultatov. Podroben opis pretvorbe in urejanja modela se nahaja v poglavju 4.2.

Definirane vstopne točke upravljanja uporabnikov so:

get /api/users Vrne registrirane uporabnike. 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. Operacija je dovoljena samo administratorju in lastniku uporabniškega računa.

post /api/uaa/user/auth/token Služi za avtentikacijo uporabnikov. Podprt je OAuth2 način 'Resource Owner Password Credentials Grant'

Definirane vstopne točke simulacije so:

post /api/tracer/jobs Ustvari novi posel za simulacijo.

put /api/tracer/materials Odstrani, doda ali posodobi privzeto listo fizikalnih lastnosti materialov.

get /api/tracer/jobs Če ima uporabnik administratorske pravice, metoda vrne vse posle. V nasprotnem primeru vrne samo posle, ustvarjene s strani dotičnega uporabnika. Rezultat je niz identifikatorjev posameznih poslov in pripadajočih statusov (na čakanju, se izvaja, preklican, končan, ...).

get /api/tracer/jobs/: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.

put /api/tracer/jobs/:id Posodobi stanje izvajanja.

put /api/tracer/jobs/:id/config Vsebina POST zahtevka vsebuje JSON z informacijami o lokaciji, polarizaciji in jakosti radijskih sevanj, ter zahtevani natančnosti in gostoti rezultatov receptorske ravnine.

put /api/tracer/jobs/:id/data Služi za nalaganje geometrijskih modelov okolice. V telesu zahtevka se pričakuje datotečni format, ki opisuje parametrizirano geometrijo v po meri izdelanem formatu in XML obliki.

put /api/tracer/jobs/:id/materials Služi za nalaganje fizikalnih lastnosti materialov, uporabljenih v geometrijskem modelu, za posamezno simulacijo. Če uporabljen material ni naveden v poslanem zahtevku in se uporablja v opisu geometrije, se uprabijo vrednosti definirane v privzetem naboru materialov. V telesu zahtevka se pričakuje lista materialov in fizikalnih lastnosti v JSON obliki.

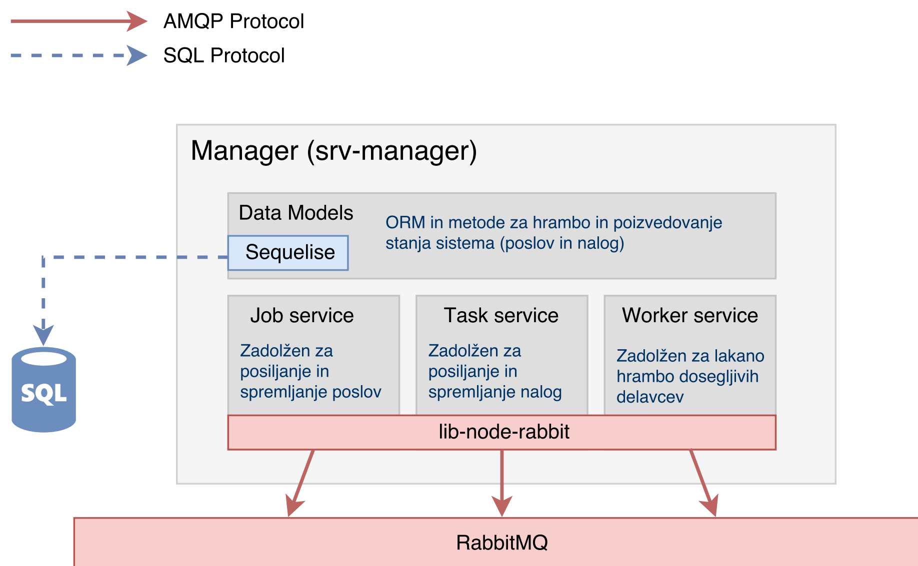
get /api/tracer/jobs/:id/results Zahtevek vrne datoteko z rezultati simulacije v XML obliki.

delete /api/tracer/jobs/:id Izbriše posel in vse pripadajoče podatke iz sistema. Če se posel izvaja, ga predhodno terminira.

Primeri uporabe spletnega grafičnega vmesnika se nahajajo v poglavju 4.3. Podroben opis parameterov vsake vstopne točke se nahaja v ločenem dokumentu programskega sklopa 2.3.

3.2 Koordiniranje nalog

Po obdelavi vhodnih podatkov zahtevka za simulacijo na aplikacijskem strežniku delo prevzame komponenta manager. Implementirana je v programskem jeziku JavaScript z orodjem NodeJS. Za komunikacijo s podatkovno bazo in podporo migracij se uporablja orodje Sequelise. Manager glede na razpoložljiva strojna sredstva, porazdeljena po posameznih instancah, razporedi naloge po sledilnikih. Informacije o razpoložljivem številu delavcev in njihovih kapacitetah pridobi ob zagonu delavca ali na zahtevo preko komunikacijskega sloja za vse delavce z razpršenim zahtevkom (ang. fan-out). Na podlagi pridobljenih informacij sestavi celotno sliko o sistemu in grafičnih kapacitetah. Podrobne informacije o vrstah komunikacije in potrjevanju zahtevkov se nahajajo v poglavju 3.4.

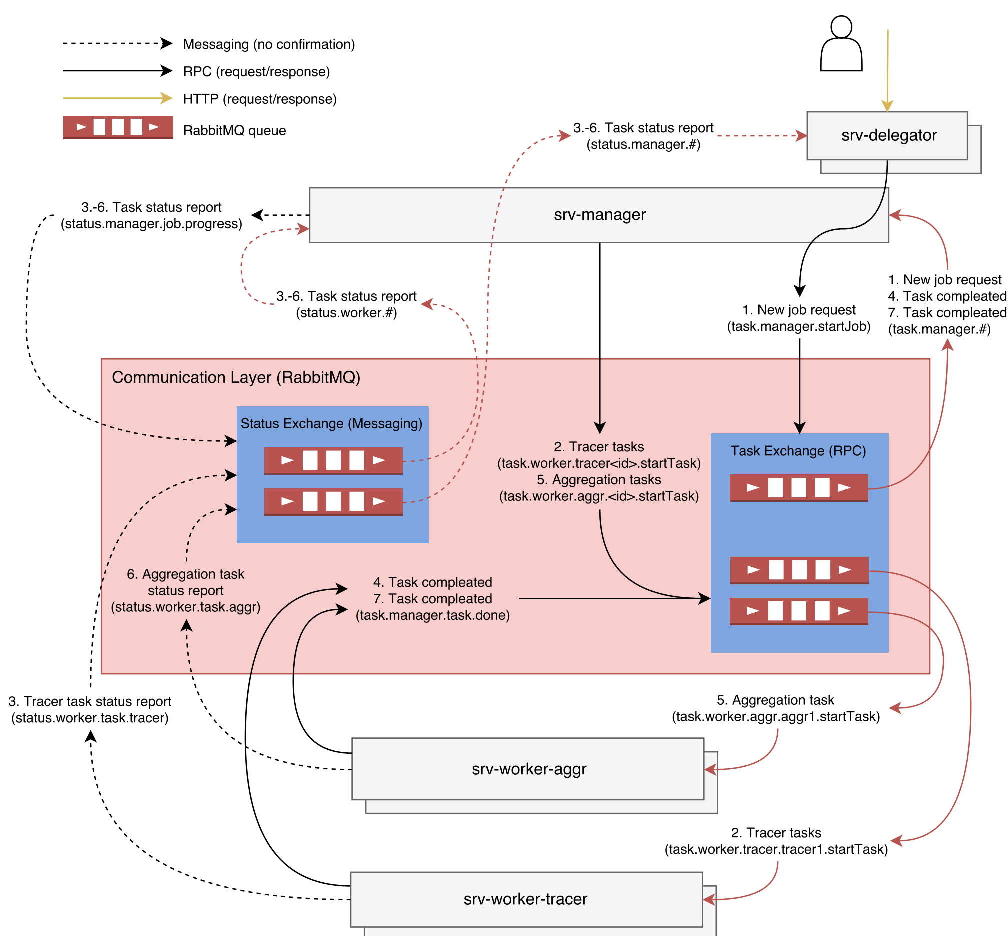


Slika 4: Diagram komponente Manager

Ob pridobitvi zahtevka za začetek izvajanja posla Manager ustvari vnos z osnovnimi podatki o izvajanu, razdeli zahtevek na večje število nalog sledilnika, shrani trenutno stanje v podatkovno bazo in vrne uspešno potrditev aplikacijskemu strežniku. Vsa stanja poslov, nalog in agregacij se shranjujejo v podatkovni bazi, katere shema je prikazana na sliki 6. Namen shranjevanja je sledljivost dogajanja, ponovna vzpostavitev delovanja komponente ob morebitnem izpadu in možnost deljenega upravljanja, kar je osnova arhitekture mikrostoritev.

Glede na obremenjenost sistema lahko Manager začasno preloži zagon posameznih delov simulacije na poznejši čas. Ob prvi poslani nalogi sledilniku

se status posla spremeni na stopnjo ‘v izvajanju’. Med izvajanjem simulacije delavci v enakomernih časovnih intervalih obveščajo Manager komponento o aktivnih nalogah in napredku. Na podlagi teh informacij komponenta istočasno sporoča napredek celotnega posla na komunikacijski sloj za eventuelno obveščanje končnih uporabnikov. Če uporabnik želi izvedeti točno stanje posla v izvajanju in pošlje poizvedbo aplikacijskemu strežniku, ta preko komunikacijske sloja povpraša Manager o podrobnem napredku posla ali vrne zadnjo pridobljeno vrednost. Ostale parametre aplikacijski sreznik pridobi direktno iz podatkovne baze in tako po nepotrebnem ne obremenjuje ostale infrastrukture.



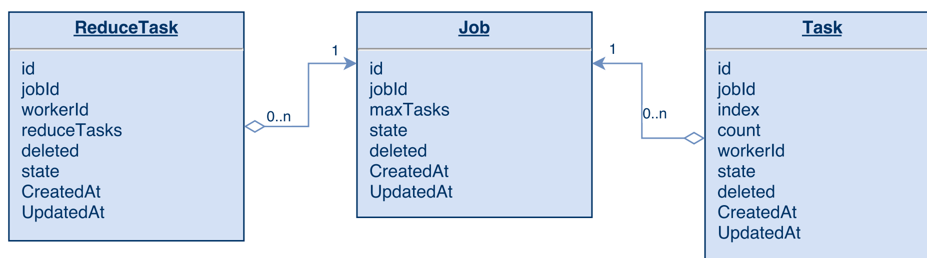
Slika 5: Shema poteka komunikacije od začteka do konca izvajanja posla

Po zaključku vseh nalog na sledilnikih Manager posodobi stanje posla na stopnjo ‘v pridobivanju rezultatov’ in ustvari zahtevek za agregacijo delnih rezultatov, katerega pravtako shrani v podatkovno bazo. Po uspešno zaklju-

čeni agregaciji Manager posodobi stanje posla na stopnjo 'končano' in objavi spremembo na komunikacijski sloj.

3.3 Podatkovna baza

Za potrebe hranjenja uporabnikov in stanja sistema zadošča že preprosta relacijska podatkovna baza. V nasem primeru smo uporabili PostgreSQL s podatkovno shemo, prikazano na sliki 6 in 7.



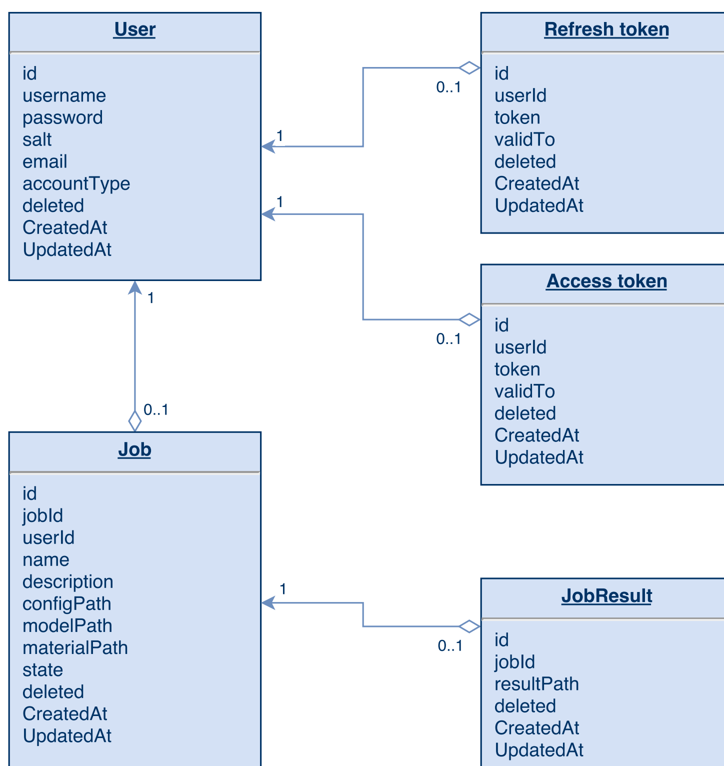
Slika 6: Shema podatkovne baze managerja (srv-manager).

Ker je sistem zgrajen na arhitekturi mikrostoritev, ima podatkovna baza pomembno vlogo hrambe stanja sistema. Posledično lahko ostale storitve migriramo, skaliramo ali jih začasno ugasnemo, brez izgube podatkov ali opravljenega dela. Za slednje je zaslužen tudi dobro zastavljen komunikacijski sloj, opisan v poglavju 3.4.. Dodatno s hrambo stanja v podatkovni bazi razbremenimo sistem in nazorno ločimo zadolžitve med integriranimi storitvami, kot je to razvidno iz podatkovnih shem na slikah 7 in 6. Tako Kompomponenta Manager kot aplikacijski strežnik vsebujta tabelo 'Posel' (ang. Job), pri čemer prva hrani samo podatke o izvajanju, druga pa le informacije, poslane s strani končnega uporabnika, in povezave do rezultatov.

3.4 Podatkovni posrednik

Podatkovni posrednik (ang. Message Broker) je ključna komponeta ART sistema za komunikacijo med komponentami. Uporabili smo že izdelano rešitev RabbitMQ, ki s premišljeno uporabo omogoča izmenjavo in filtriranje podatkov na različne načine. Na tem mestu velja poudariti, da podatkovni posrednik izrabljamo izključno kot vodilo za izmenjavo informacij in ne za prenašanje večjih količin podatkov, kot je geometrijski model, konfiguracija simulacije in opis materialov. Slednji se prenašajo preko distribuiranega datotečnega sistema NFS (ang. Network File System).

Za potrebe sistema ART izrabljamo dva načina komunikacije. Prvi spada v kategorijo sporočanja (ang. messaging) in ga uporabljamo za obveščanje



Slika 7: Shema podatkovne baze aplikacijskega strežnika (srv-delegator).

napredka nalog in poslov. Poslano sporočilo ne vsebuje naslovnika in ne zahteva potrditve prejema. Komponente, ki želijo dostopati do teh podatkov se prijavijo na izmenjevalnik (ang. Exchange), RabbitMQ pa ob pravilni konfiguraciji čakalnih vrst (ang. Queue) poskrbi za replikacijo in dostavo sporočila vsem prijavljenim storitvam. Drugi način komuniciranja je klic oddaljenih procedur ali RPC (ang. Remote Procedure Call). V tem načinu storitev pošlje zahtevek na izmenjevalnik in ustvari čakalno vrsto, preko katere pričakuje odgovor v naprej določenem časovnem okvirju. Sporočilo, enako kot v prvem primeru, ne vsebuje konkretnega naslovnika, temveč samo tip naslovne komponente in naslov čakalne vrste pošiljatelja. Na podlagi tipa prejemnika RabbitMQ preusmeri sporočilo na eno izmed prijavljenih storitev, ki zahtevek obdela in rezultat vrne pošiljatelju. Za lažje razumevanje shema 5 prikazuje uporabo komunikacijskih načinov na konkretnem primeru izvedbe enega zahtevka za simulacijo.

Za lažjo in konsistentno uporabo zgoraj opisanih metod sporočanja smo razvili namensko knjižnico (lib-node-rabbit), katero uporabljajo vse komponente sistema ART, ki se povezujejo na podatkovnega posrednika.

Interni aplikacijski vmesnik

```
{
    "userId": "user1"
    "jobId": "utyr-4ff4-kd83-myt5"
}
```

Primer 1: Zagon posla (routing: task.manager.startJob)

```
{
    "jobId": "utyr-4ff4-kd83-myt5"
}
```

Primer 2: Prekinitev posla (routing: task.manager.stopJob)

```
{
    "jobId": "utyr-4ff4-kd83-myt5",
    "taskNumber": 1,
    "taskCount": 5
}
```

Primer 3: Zagon naloge (routing: task.worker.tracer.<id>.startTask)

```
{
    "jobId": "utyr-4ff4-kd83-myt5",
    "taskNumber": 1
}
```

Primer 4: Prekinitev naloge (routing: task.worker.tracer.<id>.stopTask)

```
{
    "jobId": "utyr-4ff4-kd83-myt5",
    "timeStartedMs": 1519400236132,
    "progress": 87
}
```

Primer 5: Status posla (routing: status.manager.job.progress)

```
{
    "jobId": "utyr-4ff4-kd83-myt5",
    "taskNumber": 3,
    "timeStartedMs": 1519400236132,
    "progress": 87
}
```

Primer 6: Status naloge (routing: status.worker.task.tracer)

```
{
    "jobId": "utyr-4ff4-kd83-myt5",
    "taskNumber": 3,
    "timeStartedMs": 1519400236132,
    "timeCompletedMs": 1519400246132
}
```

Primer 7: Zaključek naloge (routing: task.manager.task.done)

```
{
    "jobId": "utyr-4ff4-kd83-myt5",
    "timeStartedMs": 1519400236132,
    "timeCompletedMs": 1519400246132
}
```

Primer 8: Zaključek posla (routing: status.manager.job.done)

3.5 Datotečni sistem

Datotečni sistem se uporablja za shranjevanje in prenašanje vhodnih podatkov posamezne simulacije, delnih in končnih rezultatov ter algoritmov za izvajanje sledenja žarkov in združevanja delnih rezultatov. Vsebina posamezne simulacije se hrani do izbrisa iz podatkovne baze. Datotečni sistem smo uporabili, ker podatkovna baza ni primerna za shranjevanje surovih binarnih podatkov in v našem primeru ne prinaša dodane vrednosti. Istočasno podatkovni posrednik ni primeren za prenašanje paketov večje velikosti, ker bi lahko prišlo do preobremenitve komunikacijskega sloja in posledično izpadov v komunikaciji med posameznimi komponentami. Iz teh razlogov smo upora-

bili distribuirani datotečni sistem NFS. Poleg vhodnih in izhodnih podatkov simulacije vsebuje tudi programe za izvajanje algoritmov, katere lahko posodobimo na vseh instancah s preprostim kopiranjem nove verzije na strežnik. Ker se za vsako simulacijo poženejo novi procesi sledilnika in agregatorja, se spremembe posodobitve odražajo šele pri novih zahtevkih. Spodaj je prikazan primer datotečne strukture.

```
/materials.xml
/config.xml
/algorithms/
/algorithms/rtx
/algorithms/aggr
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/tasks/1/
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/tasks/1/out.xml
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/tasks/2/
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/tasks/2/out.xml
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/tasks/3/
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/tasks/3/out.xml
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/materials.xml
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/config.xml
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/model.xml
/jobs/152fdc75-6d02-4e12-9885-afeeaef02ca0/result.xml
```

Primer 9: Struktura datotecnega sistema

3.6 Porazdeljeno izvajanje algoritma

Aplikacija, implementirana na Institutu Jožef Štefan, je bila prvotno izdelana v razvojnem okolju Microsoft Visual Studio in je delovala izključno na operacijskem sistemu Microsoft Windows. Po temeljiti preučitvi programske kode smo se odločili za nadgradnjo, s katero smo podprli delovanje na odprtokodnem operacijskem sistemu Linux, ki prevladuje pri ponudnikih oblačnih storitev zaradi lažjega vzdrževanja in proste dostopnosti. Predrugačeno aplikacijo smo uporabili kot sredstvo za poganjanje algoritmov sledenja žarkom, okrog katere smo razvili orodja za komuniciranje s preostalimi komponentami sistema ART. Celoto smo poimenovali sledilnik (ang. tracer), ki je sestavljen iz programske kode C++ ter upravljalca procesov in povezljivostne logike, izdelane v programskem jeziku JavaScript. Ločeni implementaciji tečeta na operacijskem sistemu kot dva ločena procesa, med katerima komunikacija poteka preko sistemskih klicev in standardnih vhodov ter izhodov.

Poleg posodobitev splošne uporabe smo predrugačili algoritem za potrebe porazdeljenega izvajanja. Omogočili smo delno izvajanje algoritma za posamezne valovne fronte in na ta način porazdelili delo med instancami sledilnikov. Posledično se čas procesiranja posamezne simulacije deli glede na razpoložljivo število grafičnih procesorjev. Pred pričetkom sledenja valovnih

front, simulirani model razdelimo na vnaprej določeno število skupin geometrijskih ravnin, ki sestavljajo celoten model. Vsako skupino ravnin dodelimo ločenemu sledilniku na način, da vsakemu sledilniku povemo skupno število nalog in zaporedno številko trenutne naloge. Iz celotne geometrije modela in zgornjih podatkov vsak sledilnik na podlagi skupne formule izlušči dodeljeno skupino ravnin. Ko algoritem sledenja žakov zazna prvo interakcijo posameznega žarka z modelom, se prepriča če zadeta ravnina pripada dodeljeni skupini. Če ne pripada, se žarek oziroma valovna fronta mora zavreči, ker bo obravnavana na drugi instanci sledilnika. Dvojna obravnava posamezne valovne fronte ni dovoljena, ker lahko neuravnoteženo prispeva h končnemu rezultatu. Po zaključku vseh procesov tipa sledilnik dobimo delne disjunktne rezultate.

3.7 Agregacija delnih rezultatov

Zadnja opracija simulacije zajema zadruževanje delnih rezultatov v končno receptorsko ravnino. Zaradi nizke obremenitve strojne opreme zadošča, da se operacija za posamezno simulacijo izvede na eni instanci komponente agregator. Algoritem se tako kot sledenje žarkov izvaja na grafičnem procesorju z uporabo programskega orodja in knjižnic CUDA, katero požene posredniška komponenta. Ta poleg nadzorovanja aktivnih procesov agregacije skrbi za komunikacijo s preostalim sistemom. Posredniška komponenta je izdelana v programskem jeziku JavaScript z uporabo orodja NodeJS. Kot smo že omenili v prejšnjem poglavju, posamezni delni rezultati vsebujejo samo izračune izbranih valovnih front. S to predpostavko lahko dobimo končni rezultat simulacije s preprostim seštevanjem realnih in imaginarnih komponent istoležnih receptorskih sfer.

4 Uporabniška navodila

Uporabniška navodila so namenjena sistemskim administratorjem, razvijalcem programske opreme ali raziskovalcem, ki želijo uporabljati sistem ART. V prvem poglavju so opisani postopki namestitve sistema, prepreostali del pa se osredotoča na izgradnjo simulacijskega modela in uporabo aplikacijskega vmesnika.

4.1 Namestitev sistema

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.

Eno izmed orodji na nameščanje, po našem mnenju primerno za potrebe projekta ART, je Ansible. Ansible uporablja datotečni format YAML za opis definicije sistema in SSH (ang. Secure Shell) za oddaljen dostop in uveljavljanje stanja.

Za namestitev moramo najprej pridobiti skripte Ansible, ki se nahajajo v repozitoriju 'project-art', in omogočiti dostop SSH do želene infrastrukture. V mapi 'environments' se nahajajo datoteke tipa 'inventory', ki definirajo dostop, dosegljivost in namen željenih vitualnih strojev. Za primer, razdelek 'manager' vsebuje naslov in prijavne podatke do virtualnega stroja, kjer želimo poganjati komponento tipa 'srv-manager'.

```
[delegator]
art-manager ansible_host=10.10.54.9 ansible_port=6033 ansible_user=root
    ansible_private_key_file=~/art/art_keys/id_rsa

[manager]
art-manager ansible_host=10.10.54.9 ansible_port=6033 ansible_user=root
    ansible_private_key_file=~/art/art_keys/id_rsa

[workers]
art-worker1 ansible_host=10.10.54.9 ansible_port=6034 ansible_user=root
    ansible_private_key_file=~/art/art_keys/id_rsa
art-worker2 ansible_host=10.10.54.9 ansible_port=6035 ansible_user=root
    ansible_private_key_file=~/art/art_keys/id_rsa
```

Primer 10: Primer namestitvene datoteke 'inventory'.

Za namestitev na testno okolje je omenjena datoteka že priložena. V primeru posodobitve sistema ali ponovne namestitve, potrebujemo namestitveni program Ansible in ključ RSA za vzpostavitev povezave SSH. Namestitev se

izvede z zagonom orodja Ansible s podano datoteko 'inventory', kot je prikazano na spodnjem primeru.

```
ansible-playbook -i environments/arctur/inventory playbook.yml
```

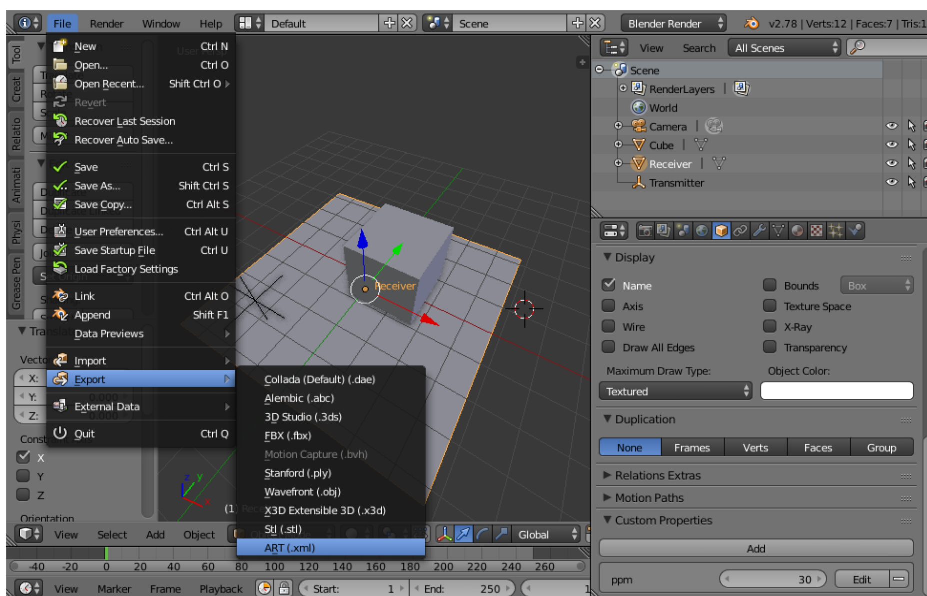
Primer 11: Zagon namestitve sistema na testno infrastrukturo.

4.2 Priprava geometrijskega modela

Izdelali smo vtičnik za odprtokodni program Blender, ki omogoča izgradnjo geometrijskega modela in pretvorbo obstoječega modela, zapisanega v enem izmed podprtih standardnih formatov, v parametriziran geometrijski model za simulacijo. Vtičnik je kompatibilen z različico Blender-ja 2.78. Poleg običajnih podatkov o prostorskih točkah in trikotnikih algoritem za simulacijo radijskih valov potrebuje dodaten opis geometrije na nivoju ravnin in konveksnih robov, kar onemogoča direktno uporabo standardnih zapisov (OBJ, 3DX, FBX, ...). Vtičnik na podlagi standardnih geometrijskih podatkov med izvozom samodejno preračuna in preoblikuje model z uporabo naprednih orodji za urejenje, vgrajenih v samo jedro Blender-ja.

Pred uporabo je potrebno vtičnik namestiti v program, kar lahko storimo preko grafičnega vmesnika ali ročno, tako da skrčeno datoteko 'io-scene-art.zip' razširimo in kopiramo vsebino v namestitveno mapo Blender-ja, zraven že obstoječih vtičnikov (<blender-mapa>/2.78/scripts/addons/). Drugi način je, da zaženemo program in izberemo meni 'File->User Preferences...'. Prikaže se modalno okno, kjer kliknemo tipko 'Install from File...'. Poiščemo in označimo datoteko 'io-scene-art.zip' na datotečnem sistemu in potrdimo izbor. Nameščeni vtičnik se prikaže na seznamu, katerega je potrebno še omogočiti s klikom na polje 'Enable'. Po uspešno naloženem vtičniku se v programskem meniju 'File->Import/Export' pojavijo dodatne možnosti za uvoz in izvoz ART modelov, kot je prikazano na sliki 8.

Blender ART vtičnik omogoča, da uporabnik uvozi ali izdela geometrijo, definira receptorsko ravnino in oddajno anteno ter vse izvozi v XML format, primeren za izvedbo simulacije. Podatki o oddajni anteni in receptorski ravnini se vnašajo s pomočjo dodatnih parametrov (ang. Custom Parameters) preko Blender vmesnika, kot je prikazano na sliki 8 v spodnjem desnem kotu. Na sliki je prikazan primer vnosa prametra 'ppm' receptorske ravnine, ki predstavlja gostoto receptorskih sfer. Za pravilno interpretacijo vrednosti morajo biti parametri poimenovani na enak način kot so definirani v aplikacijskem vmesniku. Podrobni opisi parametrov se nahajajo v tehničnem poročilu programskega sklopa 2.3. Za pravilno prepoznavo objektov mora biti oddajna antena poimenovana 'Transmitter' in receptorska ravnina 'Receiver'.



Slika 8: Izvoz geometrijskega modela

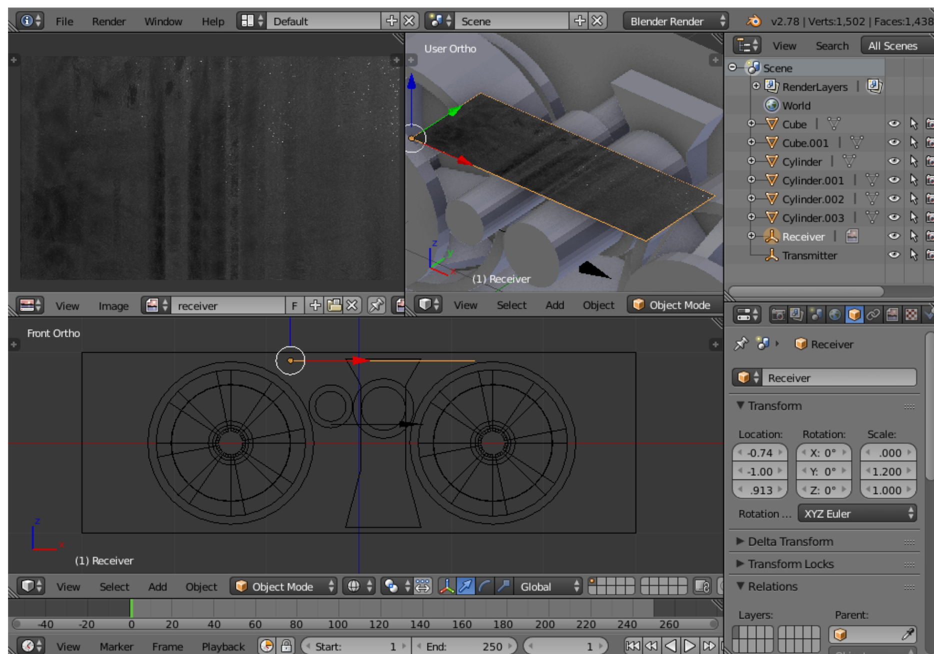
Po končanem urejanju uporabnik izvozi sceno s klikom na 'File->Export->ART(.xml)', izbere želeno mapo in vnese ime datoteke. Pridobljeno datoteko uporabi pri klicu aplikacijskega vmesnika na končno točko za pošiljanje geometrijskega modela (put /api/tracer/task/:id/data). V okviru zadnjega programskega sklopa bomo preučili tudi možnost direktne komunikacije vtičnika z aplikacijskim vmesnikom brez ročnega posredovanja uporabnika.

Po uspešno pridobljenih rezultatih simulacije lahko uporabnik s pomočjo aplikacije Blender prikaže pridobljene vrednosti receptorskih sfer v obliki sivin. Omenjen prikaz deluje samo pri rezultatih, kateri vsebujejo izgubo signala v decibelih. Za uvoz rezultatov uporabnik v aplikaciji izbere 'File->Import->ART(.xml)'. Prikaže se raziskovalec datotek s pomočjo katerega uporabnik izbere željeno datoteko, oziroma rezultat simulacije. Primer uspešno uvožene receptorske ravnine z rezultati o izgubah signala je prikazan na sliki 9.

4.3 Priprava vhodnih podatkov in izvedba simulacije

Ko imamo izdelan geometrijski model lahko pričnemo uporabljati simulacijski sistem sledenja žarkov (ART), ki se nahaja na naslovu 'http://art.xlab.si'. Aplikacijski vmesnik je dostopen na '/art', dokumentacija pa na '/docs' vstopni točki.

Pred začetkom uporabe vstopnih točk simulacije se je potrebno avtentici-



Slika 9: Prikaz izgub radijskega signala

rati z zahtevkom na 'http://art.xlab.si/art/uaa/auth/token' z uporabniškim imenom in geslom. Zahtevek v odgovoru vrne avtentikacijski žeton (ang. token), katerega je potrebno predložiti v glavi vsakega HTTP zahtevka.

```
GET /api/tracer/jobs HTTP/1.1
Host: art.xlab.si
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l
User-Agent: curl/7.58.0
Accept: */*
```

Primer 12: Primer HTTP glave zahtevka z avtentikacijskim token-om

Testno okolje trenutno uporablja osnovni način avtentikacije, pri katerem klient izdela žeton brez posredovanja strežnika na način, da združi uporabniško ime in geslo v enoten string '<username>:<password>', katerega zakodira z metodo 'base64', da pridobi avtentikacijski token. Po uspešno izvedeni avtentikaciji uporabnik izvede zahtevek za kreiranje novega posla. Zahtevek vrne identifikacijsko kodo posla, katero posredujemo pri konfiguriranju in upravljanju simulacije ter nalaganju geometrijskega modela in materialov.

```
# curl -H "Authorization: Basic <token>" -X POST http://art.xlab.si/api/tracer/jobs
{
  "id": "9a2a69ba-3cd3-4140-9b67-034a4ce94e63"
}
```

Primer 13: Primer zahtevka za kreiranje novega posla

Posel simulacije konfiguriramo tako, da podamo identifikacijsko kodo in željene parametre v json ali xml obliki. Podrobne informacije posameznih parametrov in strukture modela 'config.json' se nahajajo v tehničnem poročilu programskega sklopa 2.3.

```
# curl -d "@config.json" -H "Authorization: Basic <token>" -X POST http://art.xlab.si/api/
    tracer/jobs/9a2a69ba-3cd3-4140-9b67-034a4ce94e63/config
```

Primer 14: Primer zahtevka za konfiguracijo posla

Sledi pošiljanje geometrijskega modela in opcijsko materialov. Če posel nima podatkov o materialih se uporabijo privzete vrednosti, ki so navedene v datoteki 'materials.xml' na distribuiranem datotečnem sistemu.

```
# curl -d "@geometry.xml" -H "Authorization: Basic <token>" -X POST http://art.xlab.si/api/
    tracer/jobs/9a2a69ba-3cd3-4140-9b67-034a4ce94e63/data

# curl -d "@materials.xml" -H "Authorization: Basic <token>" -X POST http://art.xlab.si/api
    /tracer/jobs/9a2a69ba-3cd3-4140-9b67-034a4ce94e63/materials
```

Primer 15: Primer zahtevkov za pošiljanje geometrijskega modela in materialov

Po uspešno izvedeni konfiguraciji in poslanih podatkih pošljemo zahtevek za pričetek izvajanja simulacije.

```
# curl -d '{"status": "start"}' -H "Authorization: Basic <token>" -X PUT http://art.xlab.si
    /api/tracer/jobs/9a2a69ba-3cd3-4140-9b67-034a4ce94e63
```

Primer 16: Primer zahtevka za zagon simulacije

Med izvajanjem simulacija prehaja med stanji 'created, queued, running, done, canceled', kar lahko ugotovimo s poizvedbo o stanju. Stanje 'queued' pomeni, da je sistem odobril zahtevek za izvajanje posla in ga umestil na čakalno vrsto poslov, ki čakajo na proste strojne vire. Stanje 'running' pomeni, da se posel aktivno izvaja. V tem trenutku zahtevek vrača poleg stanja tudi potek simulacije v odstotkih. Na ta način lahko klient prikazuje odstotek opravljenega dela ali predvidi preostali čas izvajanja posla.

```
# curl -H "Authorization: Basic <token>" -X GET http://art.xlab.si/api/tracer/jobs/9a2a69ba
    -3cd3-4140-9b67-034a4ce94e63
{
  "id": "9a2a69ba-3cd3-4140-9b67-034a4ce94e63",
```

```
  "state": "running"
  "progress": "75.00",
}
```

Primer 17: Primer zahtevka za poizvedbo statusa posla

Če med izvajanjem simulacije posodobimo vhodne podatke in želimo trenutno izvajanje prekiniti oziroma ponovno pognati simulacijo, lahko to storimo na naslednji način, prikazan v primeru 18.

```
# curl -d '{"status": "start"}' -H "Authorization: Basic <token>" -X \
> PUT http://art.xlab.si/api/tracer/jobs/9a2a69ba-3cd3-4140-9b67-034a4ce94e63
# curl -d '{"status": "cancel"}' -H "Authorization: Basic <token>" -X \
> PUT http://art.xlab.si/api/tracer/jobs/9a2a69ba-3cd3-4140-9b67-034a4ce94e63
# curl -d '{"status": "restart"}' -H "Authorization: Basic <token>" -X \
> PUT http://art.xlab.si/api/tracer/jobs/9a2a69ba-3cd3-4140-9b67-034a4ce94e63
```

Primer 18: Primer zahtevka za zagon, ustavitev in ponovni zagon posla

Po uspešno zaključenem poslu stanje preide na stopnjo 'done' kar pomeni, da lahko izvedemo zahtevek za pridobitev rezultatov. Oblika in vsebina rezultatov je odvisna od konfiguracije in je opisana v tehničnem poročilu programskega sklopa 2.3.

```
# curl -H "Authorization: Basic <token>" -X GET http://art.xlab.si/api/tracer/jobs/9a2a69ba
    -3cd3-4140-9b67-034a4ce94e63/results
```

Primer 19: Primer zahtevka za pridobitev rezultatov posla

Za lažjo uporabo in testiranje sistema smo izdelali spletni vmesnih, ki navaja zgoraj omenjene vstopne točke skupaj s primeri zahtevkov. Preko spletnega vmesnika je možno na lažji način izvajati poizvedbe ali testirati API.

ART

ART Tracer API

tracer : Tracer operations — Show/Hide | List Operations | Expand Operations

Method	Path	Description
GET	/tracer/jobs	Get all tracer jobs
POST	/tracer/jobs	Create a new tracer job
PUT	/tracer/materials	Upload materials file
DELETE	/tracer/jobs/{jobId}	Delete and cancel job
GET	/tracer/jobs/{jobId}	Get job details
PUT	/tracer/jobs/{jobId}	Start or cancel job - set job status
PUT	/tracer/jobs/{jobId}/config	Upload job config
PUT	/tracer/jobs/{jobId}/data	Upload job model
GET	/tracer/jobs/{jobId}/results	Get job results
PUT	/tracer/jobs/{jobId}/materials	Upload materials file for job

[BASE URL: /api , API VERSION: 1.0.0]

Slika 10: Spletni vmesnik in dokumentacija API-ja

ART

ART Tracer API

tracer : Tracer operations — Show/Hide | List Operations | Expand Operations

Method	Path	Description
GET	/tracer/jobs	Get all tracer jobs
POST	/tracer/jobs	Create a new tracer job
PUT	/tracer/materials	Upload materials file
DELETE	/tracer/jobs/{jobId}	Delete and cancel job
GET	/tracer/jobs/{jobId}	Get job details

Response Class (Status 200)

Job details

Model | Example Value


```
{
  "progress": "100.00%",
  "id": "046b6c7f-0b8a-43b9-b35d-6489e6daee91",
  "state": "queued"
}
```


Response Content Type application/json ▾

Parameters

Parameter	Value	Description	Parameter Type	Data Type
jobId	(required)	ID of job to return	path	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
400	Invalid ID supplied		
404	Job not found		
default	Unexpected error		

Try it out!

Slika 11: Porženje zahtevkov preko spletnega vmesnika

5 Literatura

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>

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