Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jakob Merljak

Vmesniki za ogrodje arcControlTower

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Uroš Lotrič
Somentor: prof. dr. Andrej Filipčič

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Vmesna programska oprema igra pomembno vlogo pri izkoriščanju sistemov za visokozmogljivo računanje. Ogrodje arcControlTower, razvito za potrebe eksperimenta ATLAS, je namenjeno hkratnemu upravljanju velikega števila poslov na več gručah. Za ogrodje razvijte uporabniški vmesnik za ukazno vrstico, podoben vmesniku ogrodja ARC, in vmesnik REST, ki omogoča integracijo ogrodja v druge aplikacije.

Zahvaljujem se mentorjema izr. prof. dr. Urošu Lotriču in prof. dr. Andreju Filipčiču za strokovno pomoč, vodenje in potrpežljivost pri nastajanju diplomske naloge.

Zahvaljujem se tudi dr. Janu Joni Javoršku, mag. Barbari Krašovec, Dejanu Lesjaku in mag. Tadeju Novaku z Instituta Jožef Stefan za vse nasvete in dodatno pomoč. Prav tako se zahvaljujem kolegom na oddelku F9 Instituta Jožef Stefan, ki so mi omogočili delo v svojem laboratoriju.

Posebna zahvala gre moji družini, ki mi je omogočila študij in me ves čas podpirala in vzpodbujala.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Računalniške gruče in porazdeljeni sistemi 2

1.2 Sistemi vrst 3

1.3 Vmesna programska oprema 4

1.4 Uporaba vmesne programske opreme 4

1.5 Sistemi za upravljanje poslov 5

2 Ogrodje aCT **7**

2.1 Arhitektura ogrodja aCT 7

2.2 Vmesniki za ogrodje aCT 12

3 Implementacija uporabniškega pogona za ogrodje aCT **15**

3.1 Proces *client2arc* 16

3.2 Programski vmesnik 16

3.3 Konfiguracija, dnevniški zapisi in izjeme 19

4 Vmesnik za ukazno vrstico ogrodja aCT **21**

4.1 Implementacija 24

4.2 Overjanje uporabnikov 25

5 Vmesnik REST ogrodja aCT **27**
5.1 Implementacija strežniškega dela 29
5.2 Overjanje uporabnikov 32
5.3 Implementacija odjemalskih programov 34

6 Namestitev in uporaba ogrodja aCT **35**
6.1 Namestitev s skriptami 36
6.2 Možne izboljšave postavitve 38
6.3 Primer postavitve ogrodja aCT 39
6.4 Izkušnje s testno postavitvijo ogrodja aCT 39

7 Analiza možnih izboljšav in alternativ **41**
7.1 Izboljšave odjemalskih programov 41
7.2 Upravljanje podatkov 42
7.3 Abstrakcija nivoja podatkovne baze 43
7.4 Podpora vsebniškim tehnologijam 44
7.5 Nadzor delovanja 44
7.6 Delovni tokovi 45
7.7 Bazen povezav na bazo podatkov in primerkov objektov 45
7.8 Dodatni mehanizmi za overjanje uporabnikov 46
7.9 Primerjava z alternativami 46

8 Zaključek **49**

Literatura **52**

Povzetek

Naslov: Vmesniki za ogrodje arcControlTower

Avtor: Jakob Merljak

Obstoječa orodja za upravljanje poslov vmesne programske opreme ARC omogočajo le najbolj enostavne operacije na poslih. Z naraščanjem števila poslov se povečuje tudi zahtevnost upravljanja poslov s temi orodji. Poleg tega večina obstoječih orodij ne omogoča upravljanja poslov na več gručah hkrati. Ogrodje arcControlTower (aCT) je namenjeno učinkovitemu upravljanju velikega števila poslov na več gručah, vendar nima primernega uporabniškega vmesnika. Zato smo ogrodju najprej dodali programski vmesnik za upravljanje poslov. S programskim vmesnikom smo nato implementirali vmesnik za ukazno vrstico, ki je najbolj razširjen uporabniški vmesnik v orodjih za upravljanje poslov. Implementirali smo tudi vmesnik REST, ki omogoča strežniško postavitev ogrodja, saj zanimanje za takšno postavitev narašča. Dodani uporabniški vmesniki so nam omogočili, da smo ogrodje aCT uporabili kot orodje za upravljanje več tisoč poslov eksperimenta ATLAS.

Ključne besede: porazdeljeni sistemi, sistemi za upravljanje poslov, vmesna programska oprema.

Abstract

Title: Interfaces for arcControlTower

Author: Jakob Merljak

Existing tools for ARC-middleware job management provide only basic operations. The effort required for job management using these tools increases with the number of jobs to execute. Most of the existing tools also lack capabilities to manage jobs on multiple clusters. The arcControlTower (aCT) is a job management framework that can efficiently manage thousands of jobs over many clusters. However, it lacks a user interface that would enable its use as an alternative to other tools. Our goal was to extend aCT to enable the creation of various user interfaces. We achieved that by adding application programming interface (API) to aCT. Then, we have developed command line interface (CLI) using the API as this type of interface is the most commonly used among job management tools. Moreover, we have developed REST interface that enables a server setup of aCT since the web access to computing resources is becoming more and more popular. The user interface enabled us to use aCT for managing several thousands of jobs from the ATLAS experiment.

Keywords: distributed systems, job management systems, middleware.

Poglavje 1

Uvod

Vse od nastanka prvih računalnikov jih poskušamo ljudje uporabiti za različne naloge, ki nam olajšujejo delo ali ga v celoti opravijo namesto nas. Na začetku smo računalnike uporabljali predvsem za računanje, sčasoma odkrivamo nove načine njihove uporabe. Povečujejo se tudi potrebe po zmogljivosti računalnikov.

Kljub hitri rasti zmogljivosti procesorjev in pomnilnikov obstaja še veliko izzivov, za katere posamezen računalnik ni dovolj zmogljiv. Primeri izzivov so zahtevni izračuni in algoritmi, zapletene simulacije in obdelava zelo velikih količin podatkov. V takšnih primerih lahko uporabimo več računalnikov hkrati, ki vzporedno rešujejo dani problem. Za to so potrebni porazdeljeni računalniški sistemi in prilagojeni programi, ki znajo probleme reševati na takšnem sistemu.

Uporaba porazdeljenih sistemov se lahko izkaže za precej zapleteno. Različni sistemi se lahko tudi zelo razlikujejo, na primer po načinu dostopa. Zato se razvija programska oprema, ki uporabniku nudi univerzalen in enostaven vmesnik za uporabo najrazličnejših porazdeljenih sistemov. Ogrodje arcControlTower (aCT) [1] je primer takšne programske opreme. Na začetku je služilo za posredovanje in upravljanje računskih opravil iz sistema PanDA [2] na porazdeljene sisteme pri eksperimentu ATLAS [3]. Ogrodje je sčasoma postalo dovolj univerzalno za upravljanje opravil iz katerihkoli virov, ne samo

tistih iz sistema PanDA. Zaradi pomanjkljivosti nekaterih obstoječih orodij za upravljanje poslov smo v okviru diplomske naloge ogrodju aCT dodali vmesnike, ki omogočajo uporabo ogrodja posameznim uporabnikom na lastnih sistemih ali kot spletno storitev. Ogrodje aCT tako postane možna alternativa za druga orodja, ki omogočajo izvajanje programov na porazdeljenih sistemih.

1.1 Računalniške gruče in porazdeljeni sistemi

Računska opravila za porazdeljene sisteme prilagodimo tako, da se lahko hkrati izvajajo na več računalnikih. Nekatera opravila lahko spremenimo v več podopravil, ki se lahko izvedejo hkrati in neodvisno drugo od drugega. Takšnim opravilom pravimo tudi nerodno vzporedna (angl. embarrassingly parallel) opravila, saj jih je zelo enostavno razbiti na več podopravil in jih izvajati vzporedno. Vendar večina opravil nima takšnih lastnosti, saj so njihova podopravila medsebojno odvisna. To pomeni, da jih je potrebno natančno usklajevati. Za usklajevanje podopravil je potrebna komunikacija, zato morajo biti vsi računalniki v takšnem sistemu povezani.

Za razvoj programov, ki se vzporedno izvajajo na več računalnikih, se uporabljajo ustrezne knjižnice. Te omogočajo pisanje programa, ki se bo prenesel in vzporedno izvajal na željenem številu računalnikov. Poleg tega nudijo osnovne gradnike za identifikacijo posameznih računalnikov, na katerih tečejo primerki programa, in komunikacijo med primerki programa. Ena od takšnih knjižnic je knjižnica OpenMPI [4]. Knjižnica OpenMPI je odprtokodna implementacija standarda MPI [5], ki določa vmesnik za pošiljanje sporočil med procesi. Pošiljanje sporočil med posameznimi procesi omogoča potrebno komunikacijo in usklajevanje podopravil, zato se implementacije standarda MPI zelo pogosto uporabljajo za razvoj vzporednih programov.

Več povezanih računalnikov z nameščeno knjižnico OpenMPI in določenimi nastavitvami že predstavlja sistem, na katerem lahko poženemo vzporedne

programe. Takšne sisteme imenujemo tudi računalniške gruče (angl. cluster), posamezne računalnike v gruči pa vozlišča (angl. node). Porazdeljeni sistemi združujejo gruče, računske centre, posamezne računalnike in druge računske vire v povezan sistem. Na ta način omogočajo deljenje in boljši izkoristek računskih virov.

1.2 Sistemi vrst

Računalniške gruče po navadi uporablja več ljudi. Neka raziskovalna skupina ali laboratorij na primer vzpostavi gručo, na kateri lahko vsi poganjajo svoje programe. Če več ljudi hkrati poganja vzporedne programe, lahko hitro pride do težav, saj bodo programi zaradi večopravilnosti operacijskih sistemov tekmovali za procesorski čas, omrežno pasovno širino in vhodno-izhodne prenose. Posledično je lahko čas izvajanja programa celo daljši, kot bi bil na enem samem računalniku. Zato so potrebni mehanizmi, ki nadzorujejo izvajanje več vzporednih programov tako, da ima vsak na voljo želene zmogljivosti gruče.

Rešitev te težave je sistem vrst (angl. batch system). V sistem vrst uporabnik vstavlja posle. Posel (angl. job) določa opravilo, ki ga uporabnik želi izvesti. V okviru opravila želi uporabnik izvesti enega ali več vzporednih programov in pripraviti okolje za te programe. V grobem posel predstavlja uporabnikov program in določene parametre, kot so število vozlišč, količina pomnilnika za vsako vozlišče in dostop do grafične kartice ter drugih sistemskih virov. Uporabnik ne poganja več programa neposredno, temveč ga s parametri kot posel vloži (angl. submit) v sistem vrst. Sistem vrst posle izvaja po vrsti, vrstni red lahko določajo različni nastavljivi dejavniki. Za vsak posel sistem vrst rezervira želeno število vozlišč v gruči in na njih požene posel. Na gruči se lahko hkrati izvaja tudi več poslov, če ti ne zahtevajo vseh kapacitet gruče.

Obstaja več različnih sistemov vrst, na primer Portable Batch System [6], HTCondor [7] in SLURM [8].

1.3 Vmesna programska oprema

V okviru porazdeljenih sistemov vmesna programska oprema (angl. middleware) poenoti različne sisteme vrst na različnih operacijskih sistemih preko različnih omrežnih protokolov [9, 10]. Poleg tega vmesna programska oprema nudi mehanizme za upravljanje podatkov. Mehanizmi omogočajo prenašanje podatkov na gručo, z gruče in tudi med gručami. To se zelo pogosto uporablja, saj podatke, nad katerimi se posli izvajajo, po navadi hrani uporabnik ali podatkovni strežnik. Posli na nivoju vmesne programske opreme lahko določajo vhodne in izhodne podatke, za prenos teh nato avtomatsko poskrbi vmesna programska oprema.

Uporabnik določi posel tako, da ustvari datoteko z opisom posla. Za opisovanje poslov obstaja več jezikov, kot so JDL [11], JSDL [12] in xRSL [13]. Jeziki omogočajo določanje različnih parametrov za posel, vhodne in izhodne podatke, okolje za posel in programe ali skripte, ki naj se poženejo v okviru posla. Uporabnik posel vloži tako, da vmesni programski opremi posreduje datoteko z opisom posla.

Na nivoju vmesne programske opreme se uporabljajo tudi mehanizmi za overjanje uporabnikov. Overjanje uporabnikov predstavlja temelj za upravljanje identitet in dovoljenj, ki omogočajo boljši nadzor, varnost, določanje prioritet in deleža uporabe virov. S temi mehanizmi se lahko implementira pravilnike (angl. policy), ki omogočajo učinkovito in varno uporabo gruč ter integracijo z drugimi sistemi, recimo prej omenjenimi podatkovnimi strežniki.

ARC [14], Globus Toolkit [15, 16], gLite [17], OSG [18] so primeri implementacij vmesne programske opreme. K njim spada tudi prej omenjeni HTCondor, saj poleg mehanizmov za sistem vrst vsebuje tudi vmesnike in mehanizme vmesne programske opreme.

1.4 Uporaba vmesne programske opreme

Uporabniki pogosto uporabljajo porazdeljene sisteme preko orodij vmesne programske opreme. Ta orodja omogočajo različne operacije za upravljanje

poslov. Za izvajanje operacij se mora vsak uporabnik zaradi varnosti najprej overiti. Za overjanje uporabnikov se v implementacijah vmesne programske opreme uporabljajo certifikati standarda X.509 [19]. Ker vmesna programska oprema opravlja določena opravila za uporabnika, jo uporabnik pooblasti tako, da ustvari posredniški certifikat (angl. proxy certificate). Primer takšnega opravila je avtomatski prenos uporabnikovih podatkov na gručo ali avtomatski prenos izhodnih podatkov poslov na določeno uporabnikovo lokacijo. Osnovne operacije za upravljanje poslov so vlaganje, preverjanje stanja, pridobivanje rezultatov, čiščenje, ponovno vlaganje in ubijanje poslov. Uporabnik najprej vloži svoje posle. Nato preverja stanje poslov. Rezultate uspešno zaključenih poslov lahko prenese z gruče. Delne rezultate spodletelih poslov lahko prenese z gruče za ugotavljanje napak. Spodletele posle lahko tudi počisti ali jih ponovno vloži. Uporabnik lahko posle ubije, če se ti zataknejo ali če je kakšen parameter napačno nastavljen. Omenjene operacije so v vmesni programski opremi ARC na voljo v obliki programov *arcproxy*, *arcsub*, *arcstat*, *arcget*, *arcclean*, *arcresub* in *arckill*.

1.5 Sistemi za upravljanje poslov

Implementacije vmesne programske opreme ponujajo le najbolj enostavne operacije za upravljanje poslov [1]. Pri izvajanju velikega števila poslov se izkaže, da so ta orodja precej omejena. Če izvajanje nekaterih poslov spodleti zaradi omrežne napake ali napake na gruči, mora uporabnik to ugotoviti in razrešiti tako, da občasno preverja stanja poslov in ponovno vlaga spodletele posle. Če uporabnik posle izvaja na več gručah, mora to narediti za vsako gručo posebej. Zato obstajajo sistemi za upravljanje poslov, ki bdijo nad izvajanjem poslov in opravljajo določena opravila namesto uporabnika. Primeri takšnih opravil so ponovno vlaganje spodletelih poslov ob določenih napakah, upravljanje poslov na več gručah in razreševanje odvisnosti med posli.

Obstajajo različni sistemi za upravljanje poslov, kot sta *arcrunner* [20] in

ogrodje *aCT* [1]. Ogrodju aCT smo dodali podporo za različne uporabniške vmesnike in implementirali vmesnik za ukazno vrstico ter vmesnik REST. V naslednjih poglavjih je opisano ogrodje aCT, implementacija uporabniških vmesnikov, možna postavitev ogrodja in nekaj izkušenj s praktično uporabo ogrodja. Na koncu analiziramo možne izboljšave in primerjamo alternative z ogrodjem aCT.

Poglavje 2

Ogrodje aCT

Ogrodje aCT je sistem za upravljanje poslov. Pri uporabi ogrodja se posle ne vloži na gruče, temveč se jih vloži v ogrodje. Ogrodje nato posle vloži na gruče z uporabo vmesne programske opreme ARC.

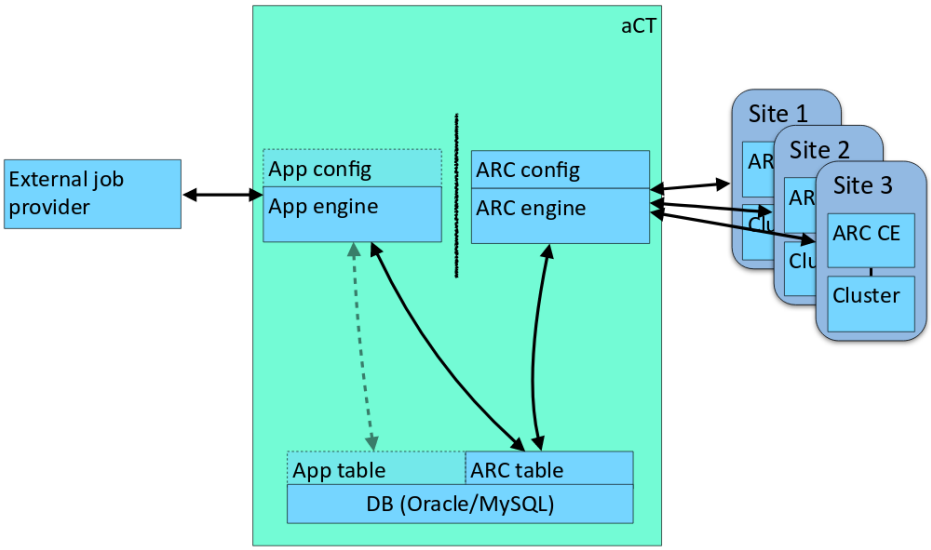
Pri vlaganju posla uporabnik določi, na katere gruče lahko ogrodje vloži posel. Ogrodje bo posel vložilo na najmanj zasedeno gručo. Ti dve lastnosti ogrodja olajšata uporabniku delo v primerjavi z orodji vmesne programske opreme. Prvič, ogrodje samo ugotavlja, kako optimalno vložiti posle, da se bodo ti izvedli kar najhitreje. Drugič, uporabniku se ni potrebno ukvarjati s prezasedenostjo gruč in ugotavljati, kdaj lahko vloži posle, saj za vlaganje na gruče poskrbi ogrodje.

Še ena prednost ogrodja aCT je, da zna ponovno vložiti spodletele posle. Lahko se na primer zgodi, da je gruča nedosegljiva zaradi omrežnih napak ali vzdrževanja. Uporabnik mora posredovati le, če posel spodleti zaradi uporabnikove napake ali zaradi daljše nedosegljivosti gruč.

2.1 Arhitektura ogrodja aCT

Ogrodje aCT deluje kot skupina več prikritih procesov (angl. daemon), ki opravljajo različna opravila na poslih. Posel v okviru ogrodja aCT je avtomat stanj. Stanje posla določa, katera opravila se lahko izvedejo na tem poslu.

Procesi periodično opravljajo določeno opravilo na poslih, nato zaspijo do izteka naslednje časovne periode. Ogrodje poženemo z zagonom glavnega procesa, *aCTMain*, ki nato ustvari in nadzoruje vse ostale procese. Če se kateri od nadzorovanih procesov zaustavi zaradi napake, ga bo *aCTMain* ponovno pognal. Procesi za delovanje in medsebojno sodelovanje potrebujejo podatkovno bazo, kjer so shranjeni podatki o vseh poslih. Ogrodje je zaradi boljše organizacije in prilagodljivosti razdeljeno v več pogonov. Vsak pogon sestavlja določen nabor procesov in tabel v podatkovni bazi. Za delovanje sta potrebna najmanj dva pogona, lahko jih je tudi več in v različnih kombinacijah. Grob pogled na arhitekturo ogrodja s stališča pogonov in njihovih interakcij lahko vidimo na sliki 2.1.



Slika 2.1: Ogrodje aCT je sestavljeno iz pogona ARC (na desni strani) in enega ali več aplikacijskih pogonov (na levi strani). Pogon ARC upravlja posle na gručah, aplikacijski pogon pa izmenjuje informacije o poslih z uporabnikom ali drugim sistemom. Aplikacijski pogoni s pogonom ARC sodelujejo tako, da dostopajo do tabel pogona ARC. Povzeto po [1].

2.1.1 Pogon ARC

Pogon ARC je obvezen del vsake kombinacije pogonov, saj opravlja vso interakcijo z gručami. Za to uporablja programski vmesnik odjemalca ARC [21]. Odjemalec ARC je odjemalska programska oprema, ki omogoča uporabo vmesne programske opreme ARC [14] s programi ali programskim vmesnikom.

Pogon ARC uporablja več tabel v podatkovni bazi:

- V tabeli *arcjobs* osvežuje podatke o poslih, ki jih vlaga na gručo ali ki jih je že vložil na gručo.
- Iz tabele *jobdescriptions* bere opise poslov v jeziku xRSL [13] v obliki nizov.
- V tabeli *proxies* hrani podatke o posredniških certifikatih uporabnikov ogrodja aCT.

ARC table na sliki 2.1 predstavlja tabelo *arcjobs*. Opisi poslov so shranjeni v ločeni tabeli zaradi optimizacije. Ker so lahko opisi poslov zelo dolgi, bi se dostop do vseh podatkov o poslih zelo upočasnil, če bi bili opisi poslov shranjeni v isti tabeli kot drugi podatki.

Pogon ARC sestavljajo štirje procesi:

- Proces *aCTSubmitter* vlaga čakajoče posle iz tabele *arcjobs* na določeno gručo.
- Proces *aCTStatus* pridobiva podatke o stanju poslov na gručah in jih osvežuje v podatkovni bazi.
- Proces *aCTFetcher* prenaša rezultate uspešno zaključenih in spodletelih poslov z gruče na začasno lokacijo.
- Proces *aCTCleaner* odstranjuje spodletele in zaključene posle z gruč in iz ogrodja aCT.

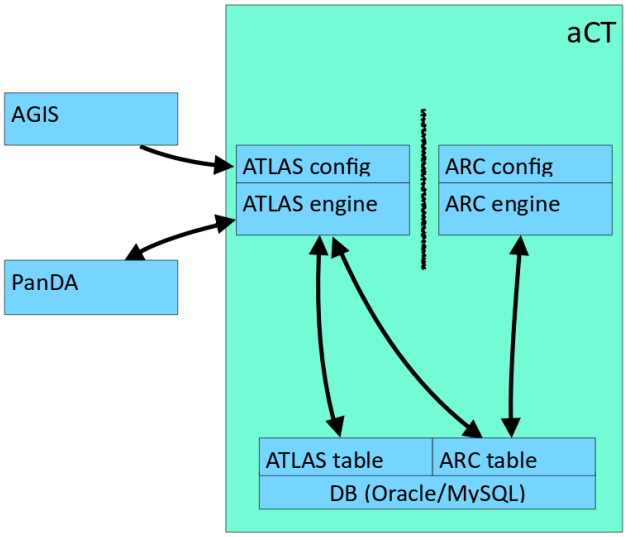
Za vsako gručo, na kateri se izvajajo posli, se zaradi večje robustnosti ustvari namenska skupina omenjenih procesov. Lahko se na primer zgodi, da ena od gruč postane nedostopna zaradi omrežnih težav. Interakcija z drugimi gručami ostane nemotena, ker se izvaja v neodvisnih ločenih procesih. Na ta način dobimo tudi enostavno paralelizacijo in skalabilnost dela z več gručami.

2.1.2 Aplikacijski pogoni

Aplikacijski pogoni opravljajo več vlog. Prva je komunikacija z uporabnikom ali sistemom za upravljanje poslov (na primer PanDA). Pri tem pogon sprejema posle od uporabnika ali sistema in posreduje informacije o poslih nazaj uporabniku ali sistemu. Druga vloga je hranjenje podatkov sprejetih poslov v podatkovni bazi. Tretja vloga je prilagoditev opisov poslov in posredovanje poslov pogonu ARC. Aplikacijski pogoni posredujejo posle pogonu ARC tako, da jih vstavljajo v tabelo *arcjobs*. Na sliki 2.1 lahko vidimo vse omenjene interakcije. Za delovanje ogrodja aCT je poleg pogona ARC potreben vsaj en aplikacijski pogon, ki omogoča vnašanje poslov v ogrodje. Lahko jih vzporedno deluje tudi več.

Primer aplikacijskega pogona je pogon ATLAS, ki se uporablja v konfiguracijah ogrodja aCT pri eksperimentu ATLAS. Pogon dobiva posle iz sistema PanDA, namenskega sistema za upravljanje poslov pri eksperimentu ATLAS. Pogon tudi periodično pošilja sistemu PanDA stanja vseh dobljenih poslov. Na sliki 2.2 lahko vidimo, kako je v ogrodje aCT vključen pogon ATLAS. V sistemu AGIS [22] so na voljo informacije o gručah in opisi virov na gručah, na katerih naj bi se posli iz sistema PanDA izvedli.

Aplikacijski pogoni morajo podatke o poslih hraniti v ločenih tabelah v podatkovni bazi. Za to obstaja več razlogov. Prvi je, da pogoni shranjujejo specifične podatke za posle in delovanje. Primer tega je pogon ATLAS, ki shranjuje dodatne podatke o poslih, potrebne za integracijo s sistemom PanDA. Ti podatki so povsem nepotrebni za vse druge posle, ki niso povezani s sistemom PanDA. Posledično uporablja pogon ATLAS ločeni tabeli *pandajobs* in *pandaarchive* za hranjenje poslov.



Slika 2.2: Aplikacijski pogon ATLAS pridobiva posle iz sistema PanDA in nazaj v sistem pošilja informacije o stanju teh poslov. Iz sistema AGIS pridobiva informacije o gručah. Pogon ATLAS hrani informacije, ki so vezane na sistem PanDA, v ločenih tabelah. Z dostopanjem do tabel pogona ARC sodeluje s pogonom ARC. Povzeto po [1].

Drugi razlog za ločene tabele je v zasnovi pogona ARC. Ta je zasnovan tako, da vse posle v tabeli *arcjobs* poskuša vložiti na gručo in nato osveževati njihovo stanje. Gruče lahko izvajajo omejeno število poslov, prav tako imajo omejeno velikost čakalne vrste za posle. Če pogonu ARC posredujemo preveč poslov, bo ta zapolnil čakalne vrste gruč. Zapolnjevanje čakalnih vrst gruč ni zaželeno, saj lahko s tem zmanjšamo prepustnost (angl. throughput) izvajanja poslov.

Ogrodje aCT zato vse posle, ki so bili vloženi v ogrodje, hrani v tabelah aplikacijskih pogonov. V tabeli *arcjobs* hrani le tiste posle, ki se izvajajo na gruči, in določeno število poslov, ki čakajo v čakalni vrsti gruče. Število poslov v čakalni vrsti vsake gruče je dovolj veliko, da ima gruča ves čas na voljo nove posle za izvajanje in so zato njene zmogljivosti dobro izkoriščene. Po drugi strani je dobro, da število poslov v čakalni vrsti gruče ni preveliko.

Število mest v čakalni vrsti ni edini dejavnik, ki odraža zasedenost gruče. Na zasedenost vplivajo tudi zahtevano število vozlišč in čas izvajanja posla. Ogrodje z ohranjanjem dovolj majhnega števila poslov v čakalnih vrstah gruč doseže, da se posli vložijo čim kasneje (angl. late binding). To poveča učinkovitost razvrščanja poslov, saj se ogrodje bolje prilagaja na zasedenost gruč.

Zagotavljanje čim večje prepustnosti izvajanja poslov spada v dobro znano družino optimizacijskih problemov razvrščanja poslov [23] (angl. JSP – Job Shop Problem). Pri ogrodju aCT je možnih parametrov in omejitev pri tem problemu veliko. Poleg gruč je potrebno upoštevati tudi na primer prioritete uporabnikov, prioritete poslov, dodatne zahteve poslov za dostop do grafične kartice, določene količine pomnilnika in namenske povezave vozlišča. Potrebno je upoštevati tudi kombinacije teh parametrov. Uporabnik ima lahko recimo dovoljenja za dostop do grafične kartice samo na nekaterih gručah.

Pogon ARC zaradi principa modularnosti ne rešuje problema razvrščanja, ukvarja se samo z izvajanjem poslov na gručah. Zato morajo biti posli, za katere še ni prostora na gručah, hranjeni ločeno v aplikacijskih pogonih. Ko je dovolj prostora v čakalnih vrstah na gručah, se lahko posreduje nove posle pogonu ARC. Ogrodje aCT je zasnovano tako, da vsak aplikacijski pogon sam razvršča posle in jih posreduje pogonu ARC. Vendar imajo pogoni pri tem omejitve, da lahko vsi pogoni pridejo do svojega deleža poslov na gručah.

2.2 Vmesniki za ogrodje aCT

Ogrodje aCT omogoča naprednejše upravljanje poslov kot večina odjemalskih orodij vmesne programske opreme. Vendar ogrodja ni bilo mogoče uporabljati kot alternativo, ker ni imelo primernega uporabniškega vmesnika. Ogrodje je potrebovalo zunanji sistem za organizacijo poslov, na primer sistem PanDA. Zato smo razvili nov pogon, ki nudi programski vmesnik za upravljanje poslov v ogrodju. S programskim vmesnikom smo nato implementirali vmesnik za ukazno vrstico in vmesnik REST. Vmesnik za ukazno

vrstico omogoča uporabniku, da si vzpostavi in uporablja ogrodje na svojem računalniku. Vmesnik REST omogoča vzpostavitev strežnika z ogrodjem aCT, s katerim se lahko preko odjemalskih programov upravlja posle. Namesto da bi uporabniki gruč uporabljali odjemalska orodja vmesne programske opreme, lahko uporabijo odjemalca za ogrodje aCT. Tega je bolj enostavno vzpostaviti, poleg tega nudi vse dodatne mehanizme za upravljanje poslov.

V naslednjih poglavjih je bolj podrobno opisana implementacija pogona in vmesnikov.

Poglavje 3

Implementacija uporabniškega pogona za ogrodje aCT

Uporabniški pogon je aplikacijski pogon, ki nudi programski vmesnik (angl. API – Application Programing Interface) za upravljanje poslov v ogrodju aCT. Ta programski vmesnik se lahko uporabi za razvoj različnih uporabniških vmesnikov.
Pogon v podatkovni bazi uporablja več tabel:

- V tabelo *clientjobs* shranjuje podatke o vseh poslih v ogrodju aCT, ki so bili vloženi preko programskega vmesnika.
- Iz tabele *arcjobs* dobiva podatke o stanju poslov na gruči.
- V tabelo *jobdescriptions* zapisuje opise poslov v jeziku xRSL.
- V tabelo *proxies* zapisuje in iz nje bere podatke o posredniških certifikatih uporabnikov.

Pogon sestavljajo tudi proces *client2arc* in moduli, ki tvorijo programski vmesnik.

3.1 Proces *client2arc*

Proces *client2arc* pogonu ARC posreduje nove posle, ko je na gručah dovolj prostora. Proces preverja zasedenost gruč tako, da pridobi število čakajočih in izvajajočih se poslov na gručah iz tabele *arcjobs*. Če je število dovolj majhno, vstavi nove posle v tabelo *arcjobs*. Trenutna implementacija posredovanja poslov je poenostavljena. Vsi uporabniki so enakovredni, za vsakega se vstavi enako število poslov. Posli se razvrščajo glede na identifikator, razvrščeni so od najmanjšega do največjega. To je najenostavnejša metoda, vendar med manj optimalnimi. Identifikatorje določi podatkovna baza glede na vrednost števca, ki se avtomatsko povečuje z vsako vstavitvijo v bazo. Ta števec se lahko spremeni ali ponastavi, kar pokvari vrstni red. Izboljšava je razvrščanje poslov glede na časovno oznako (angl. timestamp), za kar je potreben tudi dodatni indeks v podatkovni bazi na atribut s časovno oznako. Proces *client2arc* je pognan ob zagonu ogrodja aCT, požene ga glavni proces *aCTMain*. Zaključi se ob prenehanju delovanja ogrodja aCT tako, da mu proces *aCTMain* pošlje signal za zaustavitev. Če se proces med delovanjem nepričakovano zaustavi zaradi kakšne napake, ga bo proces *aCTMain* znova pognal.

3.2 Programski vmesnik

Uporabniški pogon ponuja programski vmesnik v obliki štirih modulov Python.

3.2.1 Modul *errors*

Modul *errors* definira vse izjeme (angl. exception) programskega vmesnika. Izjeme so na ta način definirane na enem mestu, kar olajša njihovo uporabo in uvažanje (angl. import) v izvorno kodo. Izjeme poleg sporočila vsebujejo tudi dodatne atribute, ki so priročni pri razreševanju ali prikazovanju težav.

3.2.2 Modul *clientdb*

Modul *clientdb* omogoča upravljanje tabele *clientjobs* v podatkovni bazi. Možne so standardne operacije nad zapisi poslov: ustvarjanje, branje, posodabljanje in brisanje (angl. CRUD – create, read, update, delete). Poleg tega obstajata operaciji za ustvarjanje in brisanje tabele, potrebni pri namestitvi ogrodja aCT. Del podatkov o poslih se nahaja tudi v tabeli *arcjobs*. Modul omogoča pridobivanje teh podatkov s povezovanjem tabel *clientjobs* in *arcjobs*. Mogoči sta dve vrsti povezovanja: notranje (angl. inner join) in levo (angl. left outer join). Notranje povezovanje vrača le podatke o tistih poslih, ki imajo zapise v obeh tabelah, torej so že na gruči oziroma so v procesu vlaganja na gručo. Za večino operacij je to zaželeno, ker se izvajajo na takšnih poslih. Pri prikazovanju stanja vseh poslov in ubijanju poslov se namesto notranjega povezovanja uporablja levo povezovanje, saj nas zanimajo tudi tisti posli, ki še niso na gruči. Takšni posli bodo pri levem povezovanju dobili nične vrednosti za atribute iz tabele *arcjobs*.

3.2.3 Modul *proxymgr*

Modul *proxymgr* omogoča upravljanje posredniških certifikatov v ogrodju aCT. Z modulom lahko pridobivamo podatke o certifikatih v dveh oblikah: datotekah in nizih. Podatke o certifikatih modul shranjuje v tabelo *proxies* in jih iz nje pridobiva. Modul lahko ustvari, bere, posodobi in briše zapise v tabeli. Za delo s certifikati in tabelo si pomaga z modulom *aCTProxy*, ki implementira večino operacij nad certifikati v ogrodju aCT.

3.2.4 Modul *jobmgr*

Modul *jobmgr* omogoča upravljanje poslov v ogrodju aCT. Za to potrebuje dostop do podatkov o poslih v podatkovni bazi in posledično uporablja modula *clientdb* in *aCTDBArc*. Slednji je del pogona ARC in ponuja vmesnik za operacije v tabeli *arcjobs*.

Operacije nad posli smo implementirali na dva načina. Prvotno se je vsaka

operacija izvedla na enem poslu naenkrat. Ta način je izjemno neučinkovit. Za vsako operacijo je najprej potrebno pridobiti določene podatke o poslu iz tabel, predvsem o stanju posla. Iz stanja posla se lahko ugotovi, ali je operacija sploh dovoljena za posel v tem stanju. Ne moremo na primer počistiti posla, ki še ni končan. Prvotno tudi nismo uporabljali povezovanja tabel, zato je pridobivanje podatkov pomenilo dve transakciji. S povezovanjem tabel smo to izboljšali na eno. Ko izvemo potrebne informacije o poslu, lahko izvedemo operacijo, kar za večino operacij pomeni spreminjanje določenih stolpcev v tabeli ali brisanje zapisov. Torej govorimo o dveh transakcijah v podatkovni bazi, pridobivanje in spreminjanje podatkov, za vsak posel posebej. Če želimo izvesti operacije na velikem številu poslov, kar je eden od namenov ogrodja aCT, se takšno število transakcij hitro izkaže za problematično.

Zato smo operacije optimizirali tako, da se izvedejo hkrati na vseh podanih poslih. Podatki o vseh poslih se hkrati preberejo, obdelajo in nato zapišejo za vse posle. Torej imamo le dve transakciji nad veliko podatki, kar lahko podatkovne baze učinkovito izvedejo. Edina pomankljivost tega načina je, da je težje uporabniku sporočiti, zakaj se operacija ni izvedla na posameznem poslu. Posli, na katerih se operacija ne more izvesti, so filtrirani že v procesu pridobivanja iz podatkovne baze. Če se operacija ni izvedla na nekaterih poslih, mora uporabnik sam pridobiti informacije o teh poslih in ugotoviti, zakaj se operacija na njih ni izvedla.

Recimo, da želi uporabnik pridobiti rezultate neke množice poslov. Nekateri od teh poslov se sploh še niso zaključili ali še niso bili vloženi na gručo. Uporabnik požene operacijo za pridobivanje rezultatov poslov in dobi rezultate le za en del želenih poslov. Če želi uporabnik ugotoviti, zakaj ni dobil rezultatov za preostale posle, mora pridobiti informacije o teh poslih, predvsem o njihovem stanju. Iz stanja poslov lahko potem ugotovi, da rezultatov za te posle še ni na voljo, ker se še niso končali.

Podrobneje velja opisati tudi operacijo za pridobivanje rezultatov poslov. Pri tej operaciji je po uspešnem prenašanju rezultatov na uporabnikovo loka-

cijo potrebno posle še izbrisati iz ogrodja. Operacija se torej izvede tako, da se odjemalcu najprej vrne podatke o lokaciji rezultatov za posle. Odjemalec mora te rezultate prenesti na želene lokacije in nato klicati operacijo čiščenja teh poslov. Pri čiščenju poslov bi lahko uporabili obstoječo standardno operacijo za čiščenje poslov, a je ta način manj učinkovit. Standardna operacija čiščenja najprej izvede branje podatkov o poslih, da ugotovi, ali se te posle sme izbrisati. Ker se je to preverilo že v prvem koraku pridobivanja rezultatov poslov (posle, katerih rezultate lahko pridobimo, lahko tudi izbrišemo), je ta korak odvečen. Za boljšo učinkovitost se zato v prvem koraku poleg lokacij rezultatov poslov vrnejo tudi identifikatorji vseh poslov, ki jih je v koraku čiščenja potrebno izbrisati. Odjemalec po uspešnem prenosu rezultatov poslov pokliče operacijo čiščenja skupaj s podatki o prenešenih poslih. Ta operacija ne poizveduje, ali je čiščenje danih poslov dovoljeno, temveč jih neposredno izbriše.

Poleg omenjenih operacij nad posli ponuja modul *jobmgr* še nekaj pomožnih funkcij, kot so preverjanje vsebine xRSL za posel, sestavljanje poti v datotečnem sistemu in pretvarjanje seznamov identifikatorjev poslov v nize.

3.3 Konfiguracija, dnevniški zapisi in izjeme

Uporabniški pogon za delovanje potrebuje nekaj parametrov iz konfiguracije pogona ARC, ki je v obliki datoteke XML. Za branje te konfiguracijske datoteke ima ogrodje aCT namenski modul *aCTConfigARC*. Potrebni parametri za delovanje uporabniškega pogona so:

- podatki za povezavo na podatkovno bazo,
- delovna mapa (angl. working directory) ogrodja aCT,
- lokacija vseh certifikatov certifikatnih avtoritet za gruče.

Gruče so v ogrodju aCT konfigurirane tako, da več gruč (ali samo ena) sestavlja eno lokacijo (angl. site). Uporabniki posle vlagajo v ogrodje aCT tako,

da določijo, na kateri lokaciji naj bi se ti posli izvedli. Lokacije se nastavlja v datoteki JSON. Uporabniški pogon potrebuje to konfiguracijo pri prejemanju poslov in predaji poslov pogonu ARC, ki mu je potrebno pretvoriti lokacijo v seznam URL-jev gruč.

Za dnevniške zapise se uporablja modul *logging* iz pythonove standardne knjižnice. Proces *client2arc* zapisuje dnevniške zapise v svojo datoteko.

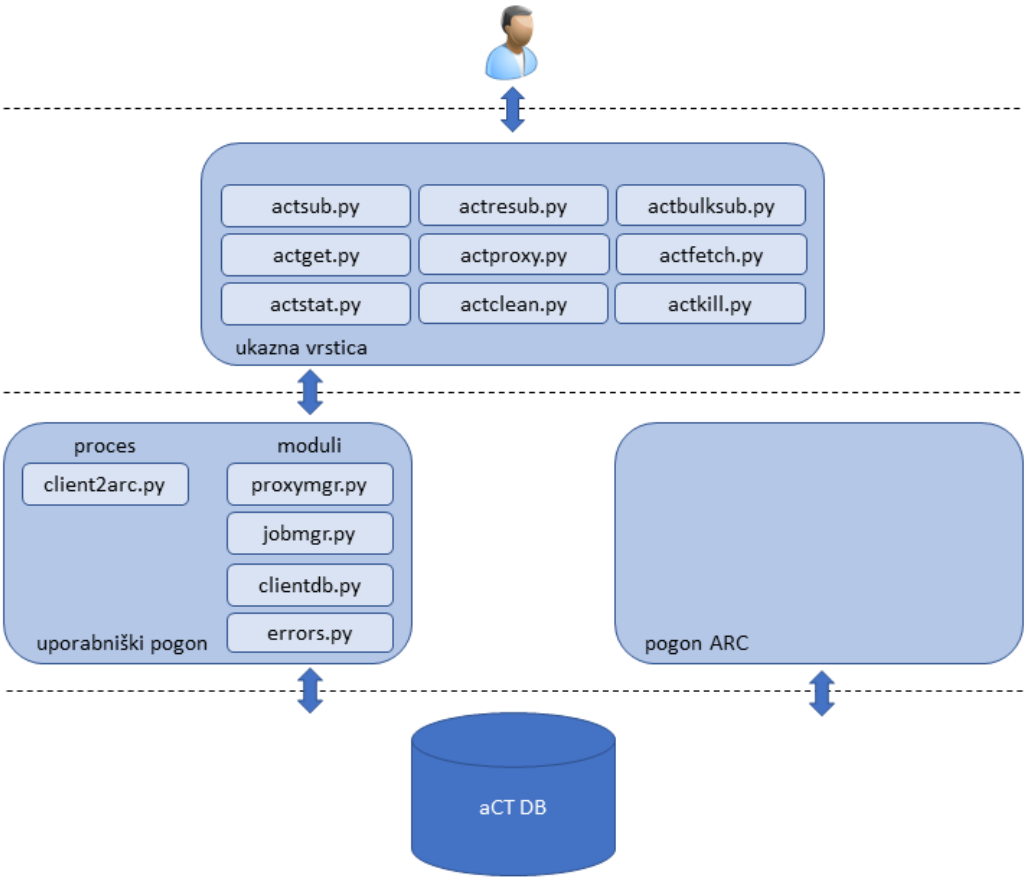
Ob napakah moduli programskega vmesnika ustvarijo dnevniški zapis in vržejo izjemo (angl. throw exception). Možne izjeme so definirane v modulu *errors* in omogočajo bolj prožno posredovanje informacij o napaki. Podobno kot moduli programskega vmesnika ravna v primeru napak tudi proces *client2arc*. Proces *client2arc* se zaključi, če nima primernega rokovalnika izjem (angl. exception handler) ali če rokovalnik ne more razrešiti problema.

Velja omeniti še obravnjavanje izjem, ki jih vrže vmesnik za podatkovno bazo MySQL [24]. Trenutno obravnavanje je poenostavljeno. Izjeme vmesnika se ulovi, ustvari se dnevniški zapis o napaki, nato se izjeme ponovno vrže (spusti skozi), da jih obravnava naslednji nivo. Ker je možnih napak v različnih situacijah veliko, bi smiselen sistem poročanja težav iz podatkovne baze MySQL bil kar obsežen in bi zahteval dodaten sloj logike za poročanje. Posledica poenostavljene implementacije je, da uporabnik v redkih primerih, če se ne drži navodil, dobi surovo izjemo namesto bolj uporabnega sporočila s težavo.

Poglavje 4

Vmesnik za ukazno vrstico ogrodja aCT

Vmesnik za ukazno vrstico sestavlja več programov za ukazno vrstico (angl. command line programs). Taka vrsta uporabniškega vmesnika je zelo prožna in hkrati relativno enostavna za implementacijo. Prožnost se kaže predvsem pri uporabi takšnega vmesnika v skriptah, s katerimi si lahko ustvarjamo poljubne delovne tokove (angl. workflow) in jih kombiniramo z bogatim naborom ostalih programov za ukazno vrstico. Še ena dobra stran takega vmesnika za ogrodje aCT je, da odjemalec ARC prav tako uporablja vmesnik za ukazno vrstico. Ker je upravljanje poslov odjemalca ARC in ogrodja aCT zelo podobno, smo lahko implementirali zelo podoben uporabniški vmesnik. Podoben vmesnik ogrodja aCT lahko zato močno olajša uporabo ogrodja vsem obstoječim uporabnikom odjemalca ARC. Umestitev vmesnika za ukazno vrstico v ogrodje je vidna na sliki 4.1.



Slika 4.1: Uporabnik posle upravlja s programi v ukazni vrstici. Ti programi omogočajo upravljanje poslov v ogrodju preko modulov uporabniškega pogona.

Uporabniški vmesnik sestavljajo naslednji programi:

- Program *actproxy* vstavi dani posredniški certifikat v podatkovno bazo ali prikaže podatke o tem certifikatu.
- Program *actsub* vstavi posel, podan kot ime datoteke xRSL, v ogrodje aCT.
- Program *actbulksub* vstavi posle, podane kot seznam datotek xRSL, v ogrodje aCT.

- Program *actstat* izpiše informacije o danih poslih.
- Program *actclean* počisti končane ali spodletele posle iz ogrodja aCT in gruč.
- Program *actget* prenese rezultate poslov iz ogrodja aCT na želeno lokacijo uporabnika.
- Program *actfetch* prenese spodletele posle z gruč. Obstaja več možnih operacij, ki jih lahko uporabnik izvede na spodletelih poslih. Lahko jih izbriše (*actclean*), prenese rezultate (*actfetch* in nato *actget*) ali ponovno vloži na gručo (*actresub*). Pri ponovnem vlaganju na gručo in brisanju je prenašanje rezultatov poslov odvečna operacija, zato ogrodje aCT pusti spodletele posle na gruči. Ogrodje bo spodletele posle preneslo z gruč le, če se uporabnik odloči prenesti posle s programom *actfetch*.
- Program *actkill* prekine izvajanje poslov v sistemu. Če posel še ni bil posredovan pogonu ARC, se posel takoj izbriše iz sistema. Sicer je potrebno operacijo ubijanja izvesti tudi na gruči. Ubijanje posla na gruči ne izbriše tega posla z gruče. Posel le preide v stanje ubitega (angl. killed). Zato je v tem primeru potrebno posel še izrecno počistiti z gruče s programom *actclean*.
- Program *actresub* ponovno vloži spodletele posle na gručo.

Iz opisa podanih programov vidimo, da so zelo podobni programom odjemalca ARC. Vendar obstaja nekaj pomembnih razlik. Prva je program *actbulksub*, ki ga ARC client ne pozna. Vstavljanje večjega števila poslov v ogrodje aCT je s tem programom bolj enostavno. Druga pomembna razlika je program *actproxy*. Ta vstavi posredniški certifikat v ogrodje aCT. Šele po tem, ko uporabnik vstavi svoj posredniški certifikat v ogrodje aCT, lahko ogrodje aCT uspešno vlaga uporabnikove posle na gručo. *actproxy* torej ni enakovreden programu *arcproxy*, ki ustvarja posredniške certifikate. Za

ustvarjanje posredniških certifikatov za uporabo v ogrodju aCT je potreben ločen program. Lahko se za to uporabi prav program *arcproxy*, ki je v vsakem primeru na voljo, saj je ogrodje aCT odvisno od odjemalca ARC, katerega del je program *arcproxy*.

Tudi program *actfetch* nima enakovrednega predstavnika v odjemalcu ARC. Odjemalec ARC upravlja posle neposredno na gručah, zato prenašanje na lokalno začasno lokacijo ni potrebno.

V primerjavi z odjemalcem ARC smo izboljšali tudi izpis podatkov o poslih (program *actstat*). Uporabnik lahko sam preko argumentov programa določa, katere podatke o poslih naj ta izpiše in v kakšnem vrstnem redu. Podatki o posameznem poslu se izpišejo v eni vrstici, kar naredi izpis veliko bolj priročen za obdelavo z drugimi programi in skriptami.

Programom smo dodali tudi argumente za filtriranje glede na ime posla ali status, kar so nam svetovali uporabniki.

4.1 Implementacija

Omenjeni programi za ukazno vrstico uporabljajo programski vmesnik uporabniškega pogona. Prva vloga teh programov je, da pretvorijo argumente iz ukazne vrstice v primerne podatkovne strukture in nato kličejo operacije, ki so na voljo v programskem vmesniku. Iz ukazne vrstice programi dobijo argumente v obliki nizov. Te je potrebno pretvoriti v vrednosti, ki jih sprejema programski vmesnik. Program lahko na primer kot argument sprejme seznam identifikatorjev poslov, na katerih naj se operacija izvede. Ta seznam je v obliki niza, identifikatorji v nizu so ločeni z vejicami. Programi morajo tak niz razbiti po vejicah na posamezne podnize, jih pretvoriti v številčne vrednosti in vstaviti v seznam. Druga vloga uporabniških programov je, da dobljene rezultate iz programskega vmesnika izpišejo uporabniku. Na primer pri povpraševanju podatkov o poslih programski vmesnik vrne seznam slovarjev s podatki. Te podatke je potrebno izpisati v obliki tabele.

Programi omogočajo tudi podroben (angl. verbose) izpis. Pri tem ustva-

rijo zapisovalnik (angl. logger) in ga nastavijo tako, da se vsi dnevniški zapisi izpisujejo na standardni izhod. Tako lahko uporabnik dostopa do vseh dnevniških zapisov, ki jih ustvarijo moduli programskega vmesnika, in dobi več informacij ob morebitnih težavah.

Ker se operacija pridobivanja rezultatov poslov izvede v dveh korakih, mora to upoštevati program *actget*. Ta zato najprej kliče funkcijo, ki vrne podatke o poslih in njihovih rezultatih. Nato prenese rezultate na uporabnikovo lokacijo. Po končanem prenosu kliče funkcijo za brisanje poslov za posle, ki jih je vrnil prvi korak.

4.2 Overjanje uporabnikov

Ogrodje aCT lahko uporablja več uporabnikov, zato je potrebno pri vsaki operaciji pridobiti identiteto uporabnika. Programi to storijo tako, da preberejo datoteko z uporabnikovim posredniškim certifikatom. Vsak program kot enega možnih argumentov sprejme lokacijo posredniškega certifikata. Če program lokacije ne dobi, poskusi prebrati certifikat s privzete lokacije. Če certifikat ni na voljo, uporabnik ne more uporabljati ogrodja aCT.

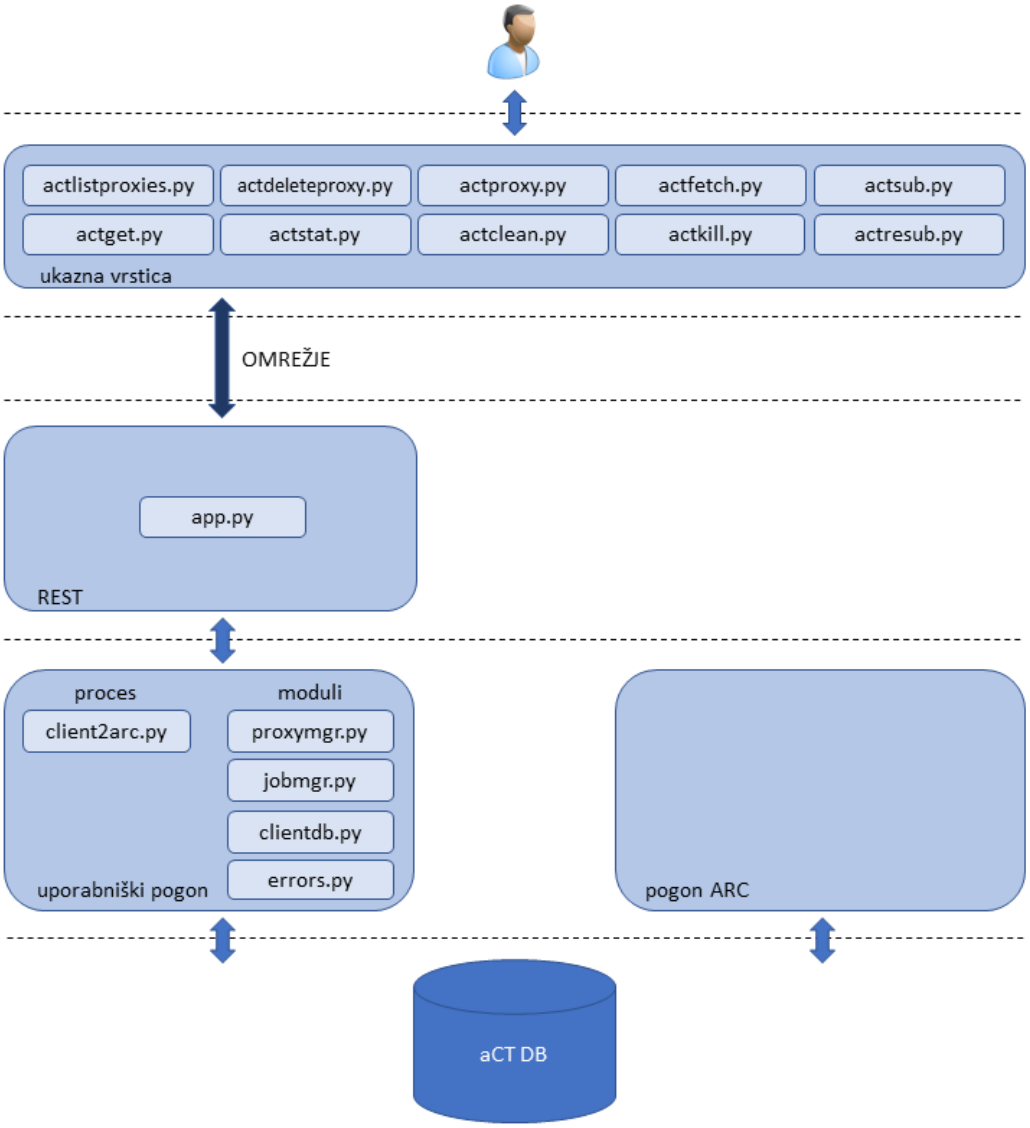
Poglavje 5

Vmesnik REST ogrodja aCT

Vmesnik REST za ogrodje aCT je zelo uporaben zaradi več razlogov. Omogoča arhitekturo odjemalec-strežnik, kar lahko močno poenostavi odjemalsko programsko opremo, ki jo potrebuje uporabnik. Vzpostavitev odjemalca ARC in ogrodja aCT je zapletena za neuke uporabnike. Arhitektura odjemalec-strežnik omogoča implementacijo enostavnega odjemalca, ki ga brez težav lahko namesti in nastavi vsak povprečen uporabnik računalnika. Bolj zapleten strežniški del namesti upravljalec strežnika.

Naslednja dobra stran vmesnikov REST je, da so zelo razširjeni in imajo dobro podporo v vseh razširjenih programskih jezikih. Tako imajo razvijalci bolj proste roke pri izbiri tehnologij, na katerih bodo temeljile njihove rešitve. Po drugi strani lahko večino vmesnikov REST uporabljamo s programi, ki podpirajo protokol HTTP. Med te spadajo vsi spletni brskalniki ali programi za ukazno vrstico, kot sta *curl* [25] ali *wget* [26]. Prav program *curl* smo uporabili pri razvoju in testiranju vmesnika. Programsko opremo, ki uporablja vmesnik REST, je tudi zelo enostavno integrirati s spletnimi aplikacijami. Vmesnik REST za ogrodje aCT torej omogoča zelo široke možnosti uporabe ogrodja in integracije ogrodja v različne sisteme ter okolja.

Na sliki 5.1 je prikazana postavitev ogrodja z vmesnikom REST, do katerega uporabnik dostopa z odjemalskimi programi preko omrežja.



Slika 5.1: Uporabnik upravlja posle s programi za ukazno vrstico. Ti programi po omrežju pošiljajo zahtevke vmesniku REST. Vmesnik REST nato želene operacije izvede preko modulov uporabniškega pogona.

5.1 Implementacija strežniškega dela

Za implementacijo vmesnika REST smo uporabili ogrodje Flask [27]. Ogrodje Flask omogoča enostaven razvoj spletnih aplikacij v jeziku Python. Spletne aplikacije, napisane v ogrodju Flask, temeljijo na objektu razreda *Flask*, ki predstavlja primerek (angl. instance) spletne aplikacije. Temu objektu lahko nato z dekoratorjem (angl. decorator) [28, 29] *route()* dodamo funkcijo, ki se izvede, ko na primerek aplikacije pride zahtevek za določen URL. Dekorator *route()* lahko določi tudi, ob katerih metodah protokola HTTP naj se funkcija izvede.

Vmesniki REST določajo vire (angl. resource) in metode, ki se lahko izvedejo na virih. V okviru vmesnika REST ogrodja aCT obstajajo trije viri:

- Vir *jobs* predstavlja posle v ogrodju aCT.
- Vir *proxies* predstavlja vse posredniške certifikate v ogrodju aCT.
- Vir *results* predstavlja rezultate poslov, ki jih lahko prenesemo iz ogrodja aCT.

Interakcija z vmesnikom REST poteka tako, da odjemalec strežniku pošilja zahtevke. Najpomembnejši deli zahtevka so metoda, URL in podatki. V URL-ju odjemalec določi, za kateri vir je podan zahtevek. Kombinacija vira in metode predstavlja operacijo, ki se lahko izvede preko vmesnika. Podatki lahko vsebujejo dodatne informacije in parametre za operacije. Tudi v URL-ju so lahko za imenom vira navedeni dodatni parametri. Glavna vloga vmesnika REST za ogrodje aCT je, da operacije vmesnika preslika v operacije programskega vmesnika uporabniškega pogona.

Zahtevek za pridobitev identifikatorja, imena in stanja vseh poslov v ogrodju aCT je prikazan na sliki 5.2. Zahtevek je sestavljen iz dveh delov, metode in URL-ja. Za to operacijo nam v zahtevek ni potrebno dati podatkov. URL lahko razdelimo na dva dela. Prvi del je */jobs*, ki določa vir, nad katerim naj se operacija izvede. Drugi del predstavlja dodatne parametre, ki bolj natančno določajo, kako naj se operacija izvede. Parametri

zahtevka na sliki 5.2 na primer določajo, naj se za vsak posel vrneta stolpca *id* in *jobname* iz tabele *clientjobs* in stolpec *arcstate* iz tabele *arcjobs*.

```
GET /jobs?client=id,jobname&arc=arcstate
```

Slika 5.2: Zahtevek za vir *jobs* z metodo *GET* in dodatnimi parametri za filtriranje podatkov iz tabel. Odgovor na zahtevek vsebuje identifikatorje, imena in stanja vseh poslov uporabnika.

Na viru *jobs* je možnih več metod.

- Metoda *DELETE* določa operacijo čiščenja poslov. V URL-ju lahko dodamo parametre za filtriranje poslov po identifikatorjih, imenu in stanju. V zahtevku za čiščenje poslov podatki niso potrebni. Vmesnik REST na zahtevek čiščenja odgovori s številom poslov, ki jih je ogrodje aCT počistilo.

- Za ponovno vlaganje in ubijanje poslov ter prenašanje spodletelih poslov se uporablja metoda *PATCH*. Podatki v zahtevku določajo, katera od teh operacij naj se izvede. Pri tej metodi je torej nujno potrebno v zahtevek dodati tudi primerne podatke. V URL-ju lahko uporabimo enake parametre za filtriranje poslov kot pri metodi *DELETE*. Odgovor na zahtevek vsebuje število poslov, na katerih se je operacija izvedla.

- Metoda *GET* se uporablja za pridobivanje informacij o poslih. Poleg že omenjenih parametrov za filtriranje omogoča še dva parametra, ki povesta, katere informacije iz tabel *arcjobs* in *clientjobs* naj se vrnejo. Podatki v zahtevku za to metodo niso potrebni. V odgovoru vmesnika so za vsak posel zahtevane informacije, strukturirane v formatu JSON.

- Vlaganje poslov v ogrodje aCT omogoča metoda *POST*. Pri tej je potrebno v podatkih zahtevka poslati datoteko z opisom posla v jeziku xRSL. Če se posel uspešno vloži, vmesnik vrne identifikator posla. Ob napaki vmesnik vrne opis napake.

Na viru *proxies* se uporabljajo naslednje metode:

- Metoda *GET* se uporablja za pridobivanje informacij o posredniških certifikatih. Trenutno niso na voljo nobeni parametri za filtriranje teh informacij. V zahtevku podatki niso potrebni. Za vsak certifikat se vrnejo informacije, strukturirane v formatu JSON.

- Metoda *DELETE* omogoča brisanje posredniških certifikatov iz ogrodja aCT. Pri tej metodi je potrebno kot parameter podati seznam identifikatorjev certifikatov, ki naj bodo izbrisani. Kot pri metodi *GET* v zahtevku podatki niso potrebni. V odgovoru dobimo število uspešno izbrisanih certifikatov.

- Posredniški certifikat lahko vstavimo v ogrodje aCT ali ga osvežimo z metodo *PUT*. Certifikat se bo osvežil, če v ogrodju že obstaja, sicer se bo vstavil v ogrodje. V odgovoru vmesnik vrne identifikator certifikata v ogrodju.

Na viru *results* je mogoča samo operacija *GET*, ki se uporablja za prenašanje rezultatov poslov. Ta operacija zahteva parameter URL *id* za identifikator posla, ki ga odjemalec želi prenesti. Pridobivanje rezultatov poslov je nekoliko bolj zahtevna operacija. Prvič, odgovor mora biti stisnjen arhiv, saj so rezultati v več datotekah in mapah. Najlažje je to storiti z orodji, ki naredijo arhiv v obliki datoteke. Python ima priročno funkcijo *shutil.make_archive*, ki ustvari arhiv iz podane mape in vseh njenih naslednikov. Drugič, ustvarjeni arhiv je potrebno izbrisati, ko odjemalec dobi rezultate. Vendar je način implementacije vmesnika REST tak, da je odgovor vrnjena vrednost funkcije, ki se odziva na zahtevek. To pomeni, da ne moremo arhiva izbrisati po tem, ko vrnemo odgovor. Možna rešitev je, da najprej ustvarimo arhiv, ga preberemo kot niz bajtov v pomnilnik, izbrišemo arhiv v obliki datoteke, in nato vrnemo niz bajtov v pomnilniku kot odgovor. Pri tem je potrebno v odgovoru nastaviti tudi format podatkov, da ga lahko odjemalec pravilno obravnava. Ker pošiljamo arhiv, kot format nastavimo *application/zip*.

5.2 Overjanje uporabnikov

Ogrodje aCT omogoča izvajanje poslov za več kot enega uporabnika, zato je potrebno overiti vsak zahtevek in ugotoviti, kateri uporabnik ga je poslal. Pri tem je najbolje uporabiti obstoječo infrastrukturo osebnih in posredniških certifikatov, saj uporabniku ni potrebno nobeno dodatno delo. Uporabi lahko kar svoj osebni certifikat, ki ga v vsakem primeru potrebuje za uporabo gruč. Overitev uporabnika z njegovim certifikatom lahko dosežemo na več načinov, najbolj primeren je z uporabo protokola TLS [30]. Ta omogoča strežniku, da zahteva overitev uporabnika, kar je potrebno pri implementaciji vmesnika REST. Poleg tega protokol omogoča še overitev strežnika in šifriranje povezave, kar je potrebno za varnost.

Uporabnik se mora za vsak zahtevek overiti s certifikatom, sicer ga strežnik zavrne. Svoj osebni certifikat mora uporabiti le za vstavljanje posredniškega certifikata v ogrodje aCT (za zahtevek PUT /proxies). Za vse ostale vrste zahtevkov mora uporabiti posredniški certifikat, ki ga ustvari s svojim osebnim certifikatom. Razlog za to je, da ima uporabnik lahko več različnih posredniških certifikatov. Da bi lahko dostopal do poslov, ki so bili vloženi z določenim certifikatom, mora uporabiti prav ta certifikat. Posli so namreč tako znotraj ogrodja aCT kot tudi na gručah vezani na certifikat, s katerim so bili vloženi.

Pri implementaciji vmesnika REST smo naleteli na kar nekaj ovir in omejitev. Ena od glavnih ovir je, da vmesniki REST v sami zasnovi nimajo stanja (so angl. stateless), uporabniških sej (angl. session), overjanja (angl. authentication) in pooblaščanja (angl. authorization). Overjanje uporabnikov je na primer nujno potrebno zaradi večuporabniške zasnove ogrodja aCT. Pooblaščanje je potrebno za omejevanje dostopa vsem, ki niso uporabniki. Odsotnost stanja in s tem uporabniških sej pomeni, da se mora vsak zahtevek overiti, kar se lahko izkaže za zelo neučinkovito. Vse te mehanizme je potrebno implemetirati z uporabo dodatnih tehnologij in dodatnega nivoja kode v odjemalcu in strežniku.

Eden od načinov overjanja uporabnikov je na primer že omenjeni način

z uporabo protokola TLS in pridobivanjem overitvenih podatkov (uporabnikovega certifikata) z nivoja protokola TLS, ki zahteva dodatno logiko v vmesniku. Ta način overjanja smo tudi uporabili.

Za overjanje in pooblaščanje na vmesnikih REST se zelo pogosto uporablja protokol OAuth 2 [31]. Implementacija protokola OAuth 2 je bolj zapletena. Protokol potrebuje overitveni strežnik, ki generira žetone (angl. token) za dostop do vmesnika. Ta del se lahko sicer poenostavi tako, da se overitveno logiko doda vmesniku. Poleg tega mora vmesnik implementirati logiko za zahtevke z žetoni in overjanje žetonov. Tudi odjemalec potrebuje dodatne zahtevke za pridobitev žetonov za dostop.

Dolgoročno gledano je zaželeno, da ogrodje aCT podpira najrazličnejše načine overitve, na primer z uporabniškimi imeni in gesli, z identiteto eduroam, protokolom Kerberos in drugimi načini. Za učinkovito implementacijo takega overjanja mora metoda overitve delovati na celotnem naboru programske opreme. Torej ne samo na ogrodju aCT, tudi na vmesni programski opremi, katere glavni namen je med drugim tudi overjanje identitet. Če bi na primer ogrodje aCT podpiralo overjanje z uporabniškimi imeni in gesli, bi morala to podpirati tudi vmesna programska oprema ARC. Trenutno ARC podpira le overjanje s certifikati. Ogrodje aCT bi sicer lahko podpiralo druge načine overitve in za uporabnike teh uporabljalo namenski certifikat za gručo, vendar je ta način zelo pomanjkljiv, ker ne omogoča več primernega nadzora nad dovoljenji in identitetami. Za primerno implementacijo poljubnega načina overjanja je torej zelo pomembna integracija z vmesno programsko opremo, ki pa trenutno še ne podpira drugih načinov overjanja. V prihajajoči verziji vmesne programske opreme, ARC 6.0, bo možna uporaba overitvenih modulov, ki bodo omogočili implementacijo poljubne overitvene metode. Ogrodje aCT bo zato lahko v prihodnosti učinkovito podpiralo različne načine overjanja uporabnikov.

5.3 Implementacija odjemalskih programov

Odjemalec za vmesnik REST ogrodja aCT je skoraj enak vmesniku za ukazno vrstico. Glavna razlika je, da odjemalski programi pošiljajo zahtevke REST na strežniški del (slika 5.1). Za pošiljanje zahtevkov se uporablja knjižnica *requests* [32]. Programi za pošiljanje zahtevkov potrebujejo več parametrov:

- naslov in vrata (angl. port) vmesnika REST,
- lokacijo uporabnikovega certifikata, potrebnega za vstavljanje posredniškega certifikata v ogrodje,
- lokacijo posredniškega certifikata, potrebnega za vse ostale operacije v ogrodju,
- lokacijo certifikata certifikatne avtoritete za overjanje strežniškega dela.

Parametre je možno nastavljati preko ukazne vrstice kot argumente programu in preko konfiguracijskih datotek. Odjemalski programi najprej prevzamejo argumente iz ukazne vrstice. Vse preostale potrebne parametre nato preberejo iz konfiguracijske datoteke, če je bila ta podana v argumentih. Manjkajoči parametri se preberejo še iz privzete konfiguracijske datoteke. Parametri, ki niso bili podani kot argumenti in tudi niso bili v konfiguracijskih datotekah, na koncu dobijo privzete vrednosti. Tak način večnivojskega nastavljanja parametrov naredi programe bolj prožne. Uporabnik lahko na primer uporablja več strežnikov aCT, v tem primeru za vsakega potrebuje ločeno konfiguracijsko datoteko, ki jo poda kot argument programu.

Poglavje 6

Namestitev in uporaba ogrodja aCT

Za vzpostavitev ogrodja aCT je potrebno namestiti in nastaviti več programov, orodij in knjižnic:

- strežnik MySQL [24] (ali MariaDB [33]),
- odjemalca ARC,
- interpreter za Python2 [34],
- knjižnico *mysql-connector* za uporabo strežnika MySQL v jeziku Python,
- komplet programskih orodij (angl. toolkit) OpenSSL [35] za overjanje certifikatov na vmesniku REST,
- knjižnico *pyOpenSSL* [36], ki omogoča dostop do orodij OpenSSL v jeziku Python,
- ogrodje *Flask*, v katerem je implementiran vmesnik REST,
- aplikacijski strežnik za poganjanje aplikacije Flask.

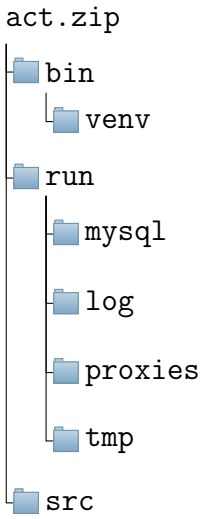
Za ogrodje aCT je na strežniku MySQL potrebno ustvariti novo bazo podatkov in uporabnika s pravicami za upravljanje te baze. Ime baze in uporabnika, geslo uporabnika in lokacijo strežnika podatkovne baze je potrebno napisati v konfiguracijsko datoteko ogrodja aCT.

Aplikacije, napisane v ogrodju Flask, je mogoče pognati na več načinov. Prvi možni način je s spletnim strežnikom, ki za izvajanje aplikacije uporablja poseben modul za jezik Python. Primer take postavitve je spletni strežnik Apache z modulom *mod_wsgi*. Drugi možen način je uporaba samostojnega, namenskega strežnika za poganjanje spletnih aplikacij, napisanih v jeziku Python. Primeri takšnih strežnikov so Gunicorn, Tornado in Twisted Web. Možna je tudi uporaba posredniških strežnikov (angl. proxy server), ki preusmerjajo zahtevke na namenski strežnik. Postavitev mora podpirati uporabo protokola TLS in overjanje odjemalcev s certifikati. Za potrebe razvoja vmesnika smo na primer uporabili razvojni strežnik ogrodja Flask s posredniškim strežnikom Nginx. Nginx overja zahtevke na nivoju protokola TLS in pridobljen uporabniški certifikat skupaj z zahtevkom preusmeri na razvojni strežnik ogrodja Flask. Takšna postavitev je potrebna, ker razvojni strežnik ne podpira pridobivanja odjemalčevega certifikata z nivoja protokola TLS.

6.1 Namestitev s skriptami

Za potrebe razvoja in testiranja ogrodja aCT smo razvili več skript, ki omogočajo enostavno, hitro in avtomatsko namestitev ogrodja z le nekaj ukazi v ukazni vrstici. Za to smo ustvarili skripto *package.sh*, ki iz izvornih datotek ustvari zapakirano datotečno hierarhijo *act.zip*, pripravljeno za poganjanje ogrodja. Datotečna hierarhija je prikazana na sliki 6.1.

Skripta *package.sh* prestavi vse izvorne datoteke programov, potrebnih za delovanje ogrodja, v mapo *bin*. Ogrodje za delovanje potrebuje mapo *log* za dnevniške zapise, mapo *tmp* za rezultate poslov in mapo *proxies* za datoteke s posredniškimi certifikati. Poleg omenjenih map potrebuje še mapo *mysql* za



Slika 6.1: V mapi *bin* se nahajajo programi ogrodja aCT. V mapi *src* se nahajo knjižnice, ki jih uporabljajo programi ogrodja. Mapo *run* uporablja ogrodje za shranjevanje podatkov med delovanjem.

svoj ločen primerek podatkovne baze. Za poganjanje ločenega primerka baze podatkov smo se odločili zato, ker je bolj enostavno avtomatizirati vzpostavitev in brisanje baze. Poleg tega uporabnik za vzpostavitev ločenega primerka ne potrebuje administratorskih pravic.

Ustvarjeni arhiv lahko nato odpakiramo v sistemu, kjer želimo vzpostaviti ogrodje. V odpakiranem arhivu je potrebno pognati skripto *setup.sh* iz mape *bin*, ki poskrbi za namestitev. Pri namestitvi se ustvari novo pythonovo navidezno okolje [37] v mapi *venv*. Navidezno okolje nam omogoča, da poganjamo programe ogrodja aCT tako, da nam ni potrebno uporabiti polne poti do njihove lokacije v datotečnem sistemu. To dosežemo tako, da navidezno okolje spremeni okoljske spremenljivke. Navidezno okolje v našem primeru nastavi okoljsko spremenljivko *PATH* tako, da ji doda pot do vseh programov ogrodja aCT. Rezultat je, da lahko programe ogrodja uporabljamo enako kot vse ostale programe, nameščene v sistemu. Navidezno okolje nam tudi omogoča nameščanje ali odstranjevanje vseh potrebnih

knjižnic brez administratorskih pravic in neodvisno od vseh ostalih knjižnic v sistemu. Namestitvena skripta *setup.sh* tudi namesti potrebne knjižnice v ustvarjeno navidezno okolje, ustvari ločen primerek podatkovne baze in vse potrebne tabele ter nastavi lokacije za dnevniške datoteke, certifikate in začasne datoteke v konfiguracijski datoteki ogrodja.

Za uporabo programov ogrodja aCT je potrebno aktivirati ustvarjeno navidezno okolje. Ogrodje se iz sistema izbriše tako, da se najprej ustavi ogrodje in bazo, deaktivira navidezno okolje in nato izbriše datotečno hierarhijo.

6.2 Možne izboljšave postavitve

Omenjeni način nameščanja in poganjanja ogrodja aCT je primeren za testiranje in eksperimentalno rabo. V večini distribucij Linux se za upravljanje programske opreme uporablja upravljalec paketov (angl. package manager). Za splošno uporabo ogrodja je potrebno ustvariti pakete ogrodja za različne upravljalce paketov. S tem se zelo poenostavi namestitev samega ogrodja in tudi vse druge programske opreme, potrebne za delovanje. Način pakiranja ogrodja je zelo odvisen od samega upravljalca paketov. Po drugi strani je način pakiranja odvisen tudi od distribucije Linux, za katero želimo ogrodje zapakirati, saj imajo različne distribucije različne politike pakiranja ter organizacije paketov. Poleg pakiranja ogrodja je za splošno rabo potrebno prilagoditi tudi način poganjanja ogrodja. Ogrodje aCT teče v obliki več prikritih procesov. V večini distribucij Linux je zaželeno, da se takšni procesi nadzorujejo in upravljajo z upravljalcem servisov (angl. service manager). Ta na primer omogoča poganjanje, ustavljanje, preverjanje stanja servisa in ponovno poganjanje servisa, če ta preneha delovati. Zelo pomembno je tudi, da lahko servisom določamo odvisnosti (angl. dependencies) na druge servise, za katere avtomatsko skrbi upravljalec servisov. Ogrodje aCT na primer za delovanje potrebuje delujoč strežnik MySQL, ki je tudi servis. Upravljalec servisov nam omogoča, da se ogrodje požene šele po tem, ko se uspešno požene servis MySQL. Način poganjanja ogrodja je odvisen od upravljalca

servisov. Nekateri upravljalci za to uporabljajo skripte, drugi konfiguracijske datoteke. Podobno kot pri pakiranju je od distribucije odvisen tudi način poganjanja servisa, ker imajo distribucije različne politike za upravljanje servisov.

6.3 Primer postavitve ogrodja aCT

Ogrodje aCT se testno uporablja za poganjanje večjega števila poslov eksperimenta ATLAS (nekaj tisoč naenkrat). Pri tej postavitvi ogrodja se uporabljajo le programi za ukazno vrstico, zato spletni vmesnik ni vzpostavljen. Ogrodje je bilo vzpostavljeno s prej omenjenimi skriptami. Na osnovi programov za ukazno vrstico je bila razvita skripta, ki omogoča enostavnejše upravljanje poslov, podobno programu *arcrunner*. Pri tem se opise in podatke poslov za vsak posel prenese v ločen imenik. Skripto se nato požene v nadimeniku. Skripta vloži vse posle v ogrodje in periodično izpisuje število čakajočih in končanih poslov ter poslov v izvajanju. Rezultate končanih poslov prenese v pripadajoče imenike. Skripta se zaključi, ko se uspešno končajo ali spodletijo vsi posli.

6.4 Izkušnje s testno postavitvijo ogrodja aCT

S testno postavitvijo ogrodja smo lahko uspešno nadomestili program *arcrunner* za vlaganje poslov eksperimenta ATLAS. Na testno postavitev se vloži približno dvesto poslov naenkrat pri vlaganju manjšega števila poslov. Ko se vlaga večje število poslov, se v ogrodje vloži okoli dvajset tisoč poslov. Čas, potreben za izvedbo vseh poslov, je skladen z zmogljivostjo gruče. Izvajanje manjšega števila poslov pri normalni zasedenosti gruče traja približno pol ure, izvajanje večjega števila poslov pa približno deset ur.

Na podlagi izkušenj s testno postavitvijo smo ogrodju dodali nekaj izboljšav. Za bolj priročno upravljanje poslov in izpisovanje informacij smo vmesniku dodali možnost filtriranja po imenih in statusu poslov. Program

za prenašanje poslov *actget* po tem, ko uspešno prenese posle, privzeto počisti posle iz ogrodja. V vmesni programski opremi ARC obstaja hrošč, zaradi katerega se v redkih primerih datoteke z rezultati ne prenesejo v celoti, kljub temu da vmesna programska oprema sporoči uspešen prenos. Trenutna rešitev tega problema je, da se rezultate poslov prenese z gruče, poslov pa se ne počisti. S tem lahko uporabnik najprej preveri rezultate. Če datoteke z rezultati niso pravilno prenesene, jih lahko uporabnik ponovno prenaša, dokler se pravilno ne prenesejo. Za to smo programu *actget* dodali dodatno stikalo, pri katerem poslov po končanem prenosu ne počisti.

Omenimo še dobro izkušnjo z robustnostjo delovanja ogrodja. Pri testnem poganjanju večjega števila poslov je nekajkrat gruča postala zelo počasna. Prišlo je do nekakšne napake v sistemu vrst SLURM, saj je zelo dolgo potreboval za čiščenje pomnilnika. Zato se je odzivnost gruče zelo upočasnila. Ogrodje aCT je kljub temu normalno delovalo naprej, le da so bile operacije z gručo počasnejše. Slaba odzivnost gruče torej ni vplivala na delovanje ogrodja, prav tako se ni izgubil noben posel ali podatek.

Poglavje 7

Analiza možnih izboljšav in alternativ

Implementirani uporabniški pogon in vmesniki zelo dobro pokrivajo vse možnosti upravljanja poslov, ki jih omogoča ogrodje aCT. S tem smo prilagodili ogrodje za samostojno ali strežniško postavitev, kar je bil naš glavni cilj. Za ogrodje obstaja več možnih izboljšav, ki presegajo zastavljene okvire diplomske naloge. V nadaljevanju analiziramo, kako je v primeru implementacije izboljšav potrebno prilagoditi uporabniški pogon in vmesnike. Za tem analiziramo še možne alternative postavitvam ogrodja aCT z uporabniškim pogonom.

7.1 Izboljšave odjemalskih programov

Med testiranjem ogrodja smo odkrili nekaj možnih izboljšav za odjemalske programe. Program *actstat* trenutno izpiše želene podatke za vsak posel. V določenih primerih bi bil priročen izpis povzetka stanja poslov, na primer za vsako stanje število poslov, ki so v tem stanju.

Trenutno program *actproxy* ne izpisuje časa veljavnosti vstavljenega posredniškega certifikata. Dodaten izpis veljavnosti bi bil dobra izboljšava programa. Poleg tega bi izpis veljavnosti uporabniku omogočil preverjanje ve-

ljavnosti že vstavljenih certifikatov.

Vmesna programska oprema ARC ima program *arccat*, ki omogoča izpisovanje standardnega izhoda posla na gruči. Z njim lahko uporabnik sledi izvajanju posla. Podoben program bi bil uporaben tudi za ogrodje aCT. Implementacija takega programa za postavitev ogrodja na lokalnem računalniku je enostavna, saj se lahko ustvari ovojnica (angl. wrapper) za program *arccat*. Implementacija takega programa za vmesnik REST je zapletenejša, saj vmesnik REST ni primeren za prenašanje tokov podatkov, kar je v tem primeru standardni izhod posla. Pri tem bi se tudi zmanjšala zmogljivost strežnika in s tem ogrodja, saj prenašanje takšnih tokov ustvarja dodaten promet.

Ogrodje aCT omogoča nastavljanje števila ponovnih poskusov za posle, če ti spodletijo. Programi za ukazno vrstico tega še ne omogočajo. Dodatno stikalo za nastavljanje števila ponovnih poskusov v programih *actsub* in *actbulksub* je tako še ena od možnih izboljšav.

7.2 Upravljanje podatkov

Ena najbolj očitnih pomanjkljivosti ogrodja je, da pri vlaganju poslov preko vmesnika REST ni mogoče vložiti nobene datoteke razen opisa posla v datoteki xRSL. To je težava, ker posel potrebuje program in podatke, na katerih naj bi se izvedel. Tudi te mora odjemalec poleg opisa posla nekako posredovati vmesni programski opremi (ne nujno ogrodju). Ogrodje trenutno ne podpira upravljanja podatkov, ki v tem primeru vključuje sprejemanje, hrambo in brisanje podatkov. Razlog za to je, da se je ogrodje aCT do sedaj uporabljalo pri eksperimentu ATLAS, kjer so vsi podatki na voljo v ločenih oddaljenih sistemih za upravljanje podatkov. Če se podatki nahajajo v oddaljenem sistemu za upravljanje ali shranjevanje podatkov, jih bo vmesna programska oprema prenesla sama od sebe. Odjemalcu ARC ali ogrodju aCT zato v tem primeru ni potrebno posredovati podatkov. Razvoj vmesnika REST omogoča uporabo ogrodja tudi uporabnikom, ki takšnih sistemov za upravljanje ali shranjevanje podatkov nimajo na voljo in posledično

ne morejo uporabljati ogrodja. Možni rešitvi za ta problem sta dve. Prva je implementacija podsistema za upravljanje podatkov znotraj ogrodja aCT. Glavna pomanjkljivost te rešitve je izjemno zapletena implementacija smiselnega sistema. Poleg tega že obstajajo rešitve za upravljanje podatkov. Zato je verjetno bolj primerna druga rešitev, pri kateri se uporabi že obstoječ sistem za upravljanje podatkov, kot sta Rucio [38] in iRods [39]. V tem primeru je potrebno poleg ogrodja aCT vzpostaviti še sistem za upravljanje podatkov in ga po možnosti integrirati v odjemalske programe. Programu *actsub* bi lahko recimo podali tudi podatkovne datoteke, ta bi jih nato posredoval v sistem za upravljanje podatkov, kjer bi bili dostopni vmesni programski opremi ARC. Omenjene težave s posredovanjem podatkovnih datotek ni, če uporabljamo lokalne odjemalske programe na sistemu, kjer teče ogrodje. Ker so v tem primeru podatkovne datoteke lokalne, jih zna avtomatsko prenesti odjemalec ARC, ko se posel vloži na gručo.

7.3 Abstrakcija nivoja podatkovne baze

Naslednja možna izboljšava ogrodja aCT, ki bi vplivala tudi na uporabniški pogon, je zamenjava orodij za dostop do baze podatkov. Ogrodje dostopa do baze podatkov MySQL preko vmesnikov za to bazo. To ni najbolj prožna rešitev, saj je potrebno za uporabo alternativnih baz podatkov kodo prilagoditi vmesnikom drugih baz. Boljša rešitev bi bila uporaba dodatnega nivoja abstrakcije podatkovne baze, kar bi omogočilo uporabo vseh razširjenih relacijskih zbirk podatkov. To za programski jezik Python omogoča zbirka programskih orodij *SQLAlchemy* [40]. Ta poleg nivoja abstrakcije omogoča tudi uporabo objektno-relacijskega preslikovalnika (angl. object-relational mapper), ki olajša uporabo podatkovnih baz v objektno usmerjenih jezikih. Ker tudi uporabniški pogon izvaja poizvedbe v podatkovni bazi, bi bilo potrebno te poizvedbe prilagoditi nivoju abstrakcije in preslikovalniku.

7.4 Podpora vsebniškim tehnologijam

Vse več gruč za izvajanje poslov uporablja vsebniške (angl. container) tehnologije. Vsebniki omogočajo vzpostavitev določenega okolja za izvajanje poslov, predvsem z vidika vse potrebne programske opreme. To olajša delo uporabnikom in sistemskim administratorjem. Uporabnikom je lažje vzpostaviti programsko opremo in knjižnice, potrebne za razvoj programov. Enostavnejša sta tudi razvoj in testiranje programov, saj lahko uporabniki posle na gručah poganjajo v popolnoma enakem okolju kot na svojih računalnikih. Tudi sistemskim administratorjem je z vsebniki lažje organizirati in upravljati okolja na gručah. Uporaba vsebnikov tudi omogoča, da sistemskim administratorjem ni potrebno vzpostavljati in vzdrževati vseh okolij, ki jih potrebujejo uporabniki. Uporabniki lahko sami ustvarijo ali prilagodijo obstoječe okolje in ga uporabijo za izvajanje poslov na gruči. Uporabniški vmesnik je potrebno za vsebniške tehnologije prilagoditi tako, da uporabniku omogoča izbiro obstoječega vsebnika ali posredovanja svojega. Možnosti je veliko, odvisne so tudi od varnostnih politik posameznega ponudnika gruče in primerne implementacije za posredovanje vsebnikov.

7.5 Nadzor delovanja

Večina programske opreme za gruče ponuja vmesnike za nadzor delovanja (angl. monitoring). Kljub temu da je nadzor delovanja domena administratorjev (tudi zaradi zasebnosti uporabnikov in varnosti sistema), so uporabnikom v splošnem na voljo vsaj osnovni podatki o gručah. Ti so predvsem število poslov v čakalni vrsti in število poslov v izvajanju. Ogrodju aCT in odjemalskim programom bi se lahko v prihodnosti dodalo možnost pridobivanja in prikazovanja osnovnih informacij o gručah in poslih. Nekaterim uporabnikom koristi tudi statistika izvajanja njihovih poslov, ki je ogrodje ne beleži. Beleženje statistike bi lahko implementirali v samem ogrodju ali v odjemalcu.

7.6 Delovni tokovi

Delovni tokovi (angl. workflow) uporabniku omogočajo ustvarjanje usmerjenih acikličnih grafov poslov, ki določajo odvisnosti med posli in tudi podatki. Če na primer posel A potrebuje podatke, ki jih ustvari posel B, mora uporabnik najprej pognati posel B in nato z rezultati posla B pognati posel A. Vse to je s trenutno programsko opremo (ne samo ogrodjem aCT) potrebno početi ročno. Uporabnik mora periodično preverjati, ali se je posel B že končal, da lahko požene posel A. Lahko se zgodi, da ima uporabnik dolgo verigo poslov, ki obdelujejo rezultate prejšnjega posla v verigi. Za vsak posel mora uporabnik ugotavljati, kdaj se konča in kdaj lahko vloži novi posel z dobljenimi rezultati. Ta postopek se lahko izkaže za zelo neučinkovitega in zapletenega, predvsem če je graf poslov zapleten ali dovolj velik. Možno je na primer, da se lahko nekateri posli ali celo verige poslov izvajajo vzporedno. Nadzorovanje takšnih grafov hitro postane počasno in neučinkovito. To se lahko razreši z razvojem dodatnega podsistema, ki odvisnosti poslov rešuje avtomatsko. Enostavnejši sistem za delovne tokove bi se lahko dodal v uporabniške programe. Pri tem bi ogrodje ostalo nespremenjeno. Uporabniški program bi vložil posel šele po tem, ko se končajo vsi potrebni posli. Druga možnost je implementacija takšnega sistema v ogrodju. Ta način je predvsem bolj učinkovit, saj bi lahko vse posle grafa vložili v ogrodje istočasno. S tem ne bi prišlo do dodatnih zamud z vlaganjem in prenašanjem podatkov.

7.7 Bazen povezav na bazo podatkov in primerkov objektov

Vmesnik REST pri vsakem zahtevku ustvari nove primerke objektov *proxymgr* in *jobmgr*, oba ustvarita še lastne primerke *clientdb*. Vsak primerek objekta *clientdb* ustvari svojo povezavo na podatkovno bazo. Vsi primerki pri vzpostavljanju začetnega stanja berejo potrebne parametre iz nastavitvene datoteke. Po končanem zahtevku se vsi primerki izbrišejo. Ta način

se izkaže za problematičnega zaradi hitrosti servisiranja zahtevka, saj branje nastavitvene datoteke in ustvarjanje primerkov ter povezav na bazo zahteva nekaj časa. Ta čas postane nezanemarljiv pri velikem številu zahtevkov in zmanjšuje zmogljivost ogrodja. Za reševanje takšnih težav se uporablja bazene (angl. pool). Ideja bazena je, da bazen na začetku ustvari več primerkov objektov ali povezav na bazo. Aplikacija ne ustvarja več novih povezav in primerkov, temveč jih pridobi iz bazena. Pri tem ni več potreben dodaten čas za ustvarjanje primerka ali branje iz nastavitvenih datotek. Aplikacija mora morda le ponastaviti nekatere atribute objekta. Ko aplikacija ne potrebuje več primerka, ga vrne v bazen. S tem tudi ni potrebnega brisanja primerka. Ta se ohrani za naslednji primer, ko aplikacija potrebuje primerek. Možna izboljšava ogrodja aCT pri preveliki režiji s primerki je torej uporaba bazena primerkov objektov. Ta se lahko v jeziku Python implementira z enostavno podatkovno strukturo, kot je vrsta.

7.8 Dodatni mehanizmi za overjanje uporabnikov

Kot smo že omenili pri overjanju uporabnikov na vmesniku REST, je v prihodnosti zaželena podpora različnih mehanizmov overjanja uporabnikov. Pri podpori novih mehanizmov bo potrebno prilagoditi uporabniški pogon in uporabniške programe, da bo uporabnik lahko izbral mehanizem in posredoval potrebne parametre. Pri vzpostavitvi sistema za upravljanje podatkov je potrebno overitvene mehanizme in uporabniški vmesnik integrirati še s tem sistemom.

7.9 Primerjava z alternativami

Ogrodje aCT je le eno od možnih orodij za upravljanje poslov na gručah. Obstajajo tudi druga orodja, ki imajo v primerjavi z ogrodjem aCT določene prednosti in slabosti. Te bomo analizirali predvsem z vidika uporabnika.

7.9.1 Odjemalec ARC

Primerjava med odjemalcem ARC in ogrodjem aCT je odvisna od postavitve ogrodja. Če uporabnik vzpostavi ogrodje na svojem računalniku, je prednost odjemalca ARC v lažji vzpostavitvi, saj vzpostavitev ogrodja aCT zahteva še namestitev baze podatkov in dodatno konfiguracijo. Če je ogrodje nameščeno na strežniku, mora uporabnik namestiti le odjemalske programe. Te je veliko enostavneje namestiti kot odjemalca ARC ali ogrodje.

V obeh primerih je ogrodje v prednosti z vidika upravljanja poslov. Edina pomanjkljivost je torej, če si uporabnik sam vzpostavi ogrodje. S predlaganimi izboljšavami namestitve ogrodja bi tudi ta pomanjkljivost skoraj izginila.

7.9.2 arcrunner

Program *arcrunner* je glavna alternativa za upravljanje poslov z vmesno programsko opremo ARC. Ko poženemo program, mu podamo enega ali več poslov. Program posle vlaga na gručo in nadzoruje njihovo izvajanje. Spodletele posle ponovno vloži, če razlog ni napaka v poslu. Uspešno končane posle prenese z gruče. Program se zaključi, ko so vsi posli uspešno končani ali spodleteli tako, da jih ni smiselno ponovno vložiti. Uporabnik mora samo pravilno podati posle in počakati, da se program konča.

V primerjavi s programi odjemalca ARC in ogrodja aCT je *arcrunner* še bolj enostaven za uporabo. Vendar ta enostavnost predstavlja tudi zelo veliko omejitev, saj *arcrunner* ne omogoča interakcije, ročnega upravljanja in ubijanja poslov ali pridobivanja podrobnejših informacij o izvajanju poslov. Za večino zahtevnejših primerov uporabe je dodaten nadzor zelo pomemben.

Programe ogrodja aCT je tudi zelo enostavno uporabiti v lastnih skriptah in jih vključiti v druge programe. Program *arcrunner* za takšen način uporabe ni primeren. Poleg tega je s programi ogrodja zelo enostavno implementirati program, podoben programu *arcrunner*. Točno takšen program se uporablja v prej omenjeni testni postavitvi. Program je omenjena skripta, ki

uporablja programe ogrodja aCT in deluje podobno kot program *arcrunner*. Skripta se bolje izkaže od programa *arcrunner* pri naraščanju števila poslov. Pri večjem številu poslov program *arcrunner* porabi nekoliko več časa da poizve in izpiše stanje poslov ob vsaki periodi. Skripta se z naraščanjem števila poslov ne upočasnjuje, saj podatke pridobiva preko programa za ukazno vrstico iz podatkovne baze. To je hitreje, ker so podatki v lokalni bazi in jih ni potrebno pridobiti z gruče.

7.9.3 Condor-G in DAGMan

Kot alternativi lahko omenimo tudi sistema Condor-G in DAGMan.

Condor-G [16, 41] je sistem, ki z uporabnikovega vidika opravlja zelo podobne vloge kot ogrodje aCT, le da posle vlaga na gruče Condor. DAGMan [42] je sistem, ki omogoča določanje odvisnosti med posli in ustvarjanje usmerjenih acikličnih grafov iz poslov. Sistem avtomatsko posle razvrsti tako, da se izvedejo v pravilnem vrstnem redu. To nam omogoča avtomatizacijo prej omenjenih delovnih tokov. Prednost sistema DAGMan v primerjavi z ogrodjem aCT so torej dodatne možnosti pri upravljanju poslov. DAGMan je sicer eno od orodij programske opreme HTCondor.

Bolj podrobna analiza prednosti in slabosti omenjenih sistemov v primerjavi z ogrodjem aCT bi zahtevala tudi analizo implementacij sistemov in analizo razlik med programsko opremo HTCondor in ARC, kar je zunaj našega okvira. Z uporabnikovega vidika je uporaba sistema Condor-G zelo podobna, medtem ko DAGMan predstavlja dodatno orodje za izračunavanje grafov in razvrščanje poslov. Ogrodje aCT takšnega orodja nima, kar lahko štejemo kot pomanjkljivost za nekatere uporabnike.

Poglavje 8

Zaključek

Obstoječa orodja za upravljanje poslov z vmesno programsko opremo ARC so precej pomanjkljiva. Ogrodje aCT omogoča naprednejše upravljanje poslov, vendar za ogrodje ni obstajal uporabniški vmesnik, saj se ogrodje uporablja predvsem za upravljanje poslov pri eksperimentu ATLAS, kjer uporabniški vmesnik ni potreben. Zato smo v okviru diplomske naloge ogrodju aCT dodali podporo za razvoj uporabniških vmesnikov in razvili dva uporabniška vmesnika.

Ogrodju smo najprej dodali nov pogon, ki izpostavlja programski vmesnik za upravljanje poslov ogrodja. Programski vmesnik nam je omogočil razvoj različnih uporabniških vmesnikov. Razvili smo vmesnik za ukazno vrstico, saj je ta med orodji za upravljanje poslov najbolj razširjen. Vmesnik za ukazno vrstico je zelo soroden orodjem vmesne programske opreme ARC.

Pri uporabi porazdeljenih sistemov postajajo spletni vmesniki vedno bolj razširjeni, zato smo ogrodju dodali tudi vmesnik REST in ga na ta način prilagodili za strežniško postavitev. Pri tem smo naleteli na nekaj ovir pri overjanju uporabnikov, potrebno je bilo smiselno posredovati uporabnikove certifikate vmesniku. Za vmesnik REST smo razvili tudi odjemalske programe, ki so podobni programom vmesnika ukazne vrstice. Razlikujejo se le po načinu komunikacije z ogrodjem.

Ogrodje je uspešno nadomestilo program *arcrunner* za poganjanje večjega

števila poslov. Na podlagi izkušenj s testno postavitvijo ogrodja smo naredili nekaj popravkov in dodali izboljšave ogrodju ter odjemalskim programom. Uspešna izkušnja s testno postavitvijo ogrodja kaže, da nam je ogrodje uspelo primerno prilagoditi za uporabniške vmesnike.

Možnih je še več izboljšav tako ogrodja kot tudi uporabniških vmesnikov. Izboljša se lahko način namestitve ogrodja, ki bi uporabnikom olajšal vzpostavitev ogrodja. V ogrodju so možne dodatne optimizacije, ki bi povečale učinkovitost in zmogljivost ogrodja. Z uvajanjem vsebniških tehnologij in novih načinov overjanja uporabnikov v porazdeljene sisteme se odprejo zanimive možnosti za integracijo teh tehnologij v ogrodje. S tem bi ogrodje omogočalo dodatne poenostavitve uporabniku. Z uvedbo alternativnih tehnologij overjanja bi na primer marsikateremu uporabniku prihranili nevšečnosti s certifikati. Z integracijo tehnologije vsebnikov bi lahko nudili večji nadzor nad okoljem na gruči in s tem olajšali delo uporabnikom in jim ponudili dodatne možnosti za razvoj vzporednih programov.

Literatura

[1] J. Nilsen, D. Cameron, and A. Filipčič, "ARC Control Tower: A flexible generic distributed job management framework," *Journal of Physics: Conference Series*, vol. 664, no. 6, p. 062042, 2015.

[2] K. De, A. Klimentov, T. Maeno, P. Nilsson, D. Oleynik, S. Panitkin, A. Petrosyan, J. Schovancova, A. Vaniachine, and T. Wenaus, "The future of PanDA in ATLAS distributed computing," *Journal of Physics: Conference Series*, vol. 664, no. 6, p. 062035, 2015.

[3] T. A. Collaboration, "The ATLAS experiment at the CERN Large Hadron Collider," *Journal of Instrumentation*, vol. 3, no. 08, p. S08003, 2008.

[4] "Open MPI: Open Source High Performance Computing." Dosegljivo: `https://www.open-mpi.org/`. [Dostopano 11. 4. 2018].

[5] "MPI: A Message-Passing Interface Standard Version 3.1." Dosegljivo: `http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`. [Dostopano 11. 4. 2018].

[6] R. L. Henderson, "Job scheduling under the portable batch system," in *Job Scheduling Strategies for Parallel Processing* (D. G. Feitelson and L. Rudolph, eds.), (Berlin, Heidelberg), pp. 279–294, Springer Berlin Heidelberg, 1995.

[7] "HTCondor High Throughput Computing." Dosegljivo: http://research.cs.wisc.edu/htcondor/description.html. [Dostopano 9. 5. 2018].

[8] "slurm workload manager." Dosegljivo: https://slurm.schedmd.com/quickstart.html. [Dostopano 9. 5. 2018].

[9] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Prentice-Hall, Inc., 2002.

[10] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*. Addison-Wesley, 2005.

[11] "Job Description Language JDL." Dosegljivo: https://www.ogf.org/documents/GFD.56.pdf. [Dostopano 15. 6. 2018].

[12] "Job Submission Description Language (JSDL) Specification, Version 1.0." Dosegljivo: https://www.ogf.org/documents/GFD.56.pdf. [Dostopano 15. 6. 2018].

[13] NorduGrid, "Extended Resource Specification Language." Dosegljivo: http://www.nordugrid.org/documents/xrsl.pdf, 2017. [Dostopano 12. 12. 2017].

[14] NorduGrid, "Nordugrid Advanced Resource Connector." Dosegljivo: http://www.nordugrid.org/arc/about-arc.html, 2018. [Dostopano 10. 5. 2018].

[15] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," in *Network and Parallel Computing* (H. Jin, D. Reed, and W. Jiang, eds.), (Berlin, Heidelberg), pp. 2–13, Springer Berlin Heidelberg, 2005.

[16] F. Berman, G. C. Fox, and A. J. G. Hey, *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.

[17] E. Laure, F. Hemmer, F. Prelz, S. Beco, S. Fisher, M. Livny, L. Guy, M. Barroso, P. Buncic, P. Z. Kunszt, A. Di Meglio, A. Aimar, A. Edlund, D. Groep, F. Pacini, M. Sgaravatto, and O. Mulmo, "Middleware for the next generation Grid infrastructure," no. EGEE-PUB-2004-002, p. 4 p, 2004.

[18] P. Avery, "Open science grid: Building and sustaining general cyberinfrastructure using a collaborative approach," *First Monday*, vol. 12, no. 6, 2007.

[19] "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." Dosegljivo: `https://www.ietf.org/rfc/rfc5280.txt`. [Dostopano 19. 6. 2018].

[20] "Using arcrunner to run large job sets in FGCI." Dosegljivo: `https://research.csc.fi/fgci-using-arcrunner-to-run-large-job-sets`. [Dostopano 10. 5. 2018].

[21] NorduGrid, "ARC Clients." Dosegljivo: `http://www.nordugrid.org/documents/arc-ui.pdf`, 2017. [Dostopano 13. 12. 2017].

[22] A. Anisenkov, A. D. Girolamo, A. Klimentov, D. Oleynik, A. Petrosyan, and the Atlas Collaboration, "AGIS: The ATLAS Grid Information System," *Journal of Physics: Conference Series*, vol. 513, no. 3, p. 032001, 2014.

[23] J. Carlier and E. Pinson, "An algorithm for solving the job-shop problem," *Management Science*, vol. 35, no. 2, pp. 164–176, 1989.

[24] "MySQL: The world's most popular open source database." Dosegljivo: `http://docs.python-requests.org/en/master/`. [Dostopano 26. 2. 2018].

[25] "curl: command line tool and library for transferring data with URLs." Dosegljivo: `https://curl.haxx.se/`. [Dostopano 13. 6. 2018].

[26] "GNU Wget." Dosegljivo: https://www.gnu.org/software/wget/#content. [Dostopano 13. 6. 2018].

[27] "Flask - web development, one drop at a time." Dosegljivo: http://flask.pocoo.org/. [Dostopano 26. 2. 2018].

[28] B. Slatkin, *Effective Python: 59 Specific Ways to Write Better Python*. Addison-Wesley, 2015.

[29] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2015.

[30] "The Transport Layer Security (TLS) Protocol Version 1.2." Dosegljivo: https://tools.ietf.org/html/rfc5246. [Dostopano 13. 6. 2018].

[31] "The OAuth 2.0 Authorization Framework." Dosegljivo: https://tools.ietf.org/html/rfc6749. [Dostopano 16. 5. 2018].

[32] "Requests: HTTP for Humans." Dosegljivo: http://docs.python-requests.org/en/master/. [Dostopano 26. 2. 2018].

[33] "Mariadb." Dosegljivo: https://mariadb.org/. [Dostopano 13. 6. 2018].

[34] "Python." Dosegljivo: https://www.python.org/. [Dostopano 13. 6. 2018].

[35] "OpenSSL cryptography and SSL/TLS toolkit." Dosegljivo: https://www.openssl.org/. [Dostopano 13. 6. 2018].

[36] "pyopenssl." Dosegljivo: https://pyopenssl.org/en/stable/. [Dostopano 13. 6. 2018].

[37] "Virtual Environments and Packages." Dosegljivo: https://docs.python.org/3/tutorial/venv.html. [Dostopano 15. 3. 2018].

[38] "rucio." Dosegljivo: https://rucio.cern.ch/. [Dostopano 26. 6. 2018].

[39] "iRods." Dosegljivo: https://irods.org/. [Dostopano 26. 6. 2018].

[40] "SQLAlchemy: The Python SQL Toolkit and Object Relational Mapper." Dosegljivo: https://www.sqlalchemy.org/. [Dostopano 21. 3. 2018].

[41] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, pp. 237–246, Jul 2002.

[42] "HTCondor Version 8.7.7 Manual." Dosegljivo: http://research.cs.wisc.edu/htcondor/manual/latest/. [Dostopano 19. 4. 2018].