UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jakob Merljak

Računalništvo v oblaku

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Marko Bajec

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01559/2009
Datum: 05.04.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JAKOB MERLJAK**

Naslov: **RAČUNALNIŠTVO V OBLAKU**
CLOUD COMPUTING

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Tako imenovano »računalništvo v oblaku« je vedno bolj priljubljeno področje računalništva. Osnovna ideja, ki se skriva za omenjenim pojmom, je ponuditi visoko zmogljive, zanesljive, prilagodljive in cenovno ugodne računalniške storitve, ki bodo uporabnikom dostopne preko enostavnih spletnih vmesnikov oziroma prek spleta.

Kandidat naj v svojem diplomskem delu predstavi omenjeno področje, našteje zgodovinske dogodke, ki so pripeljali do razvoja omenjenega pristopa, opiše ključne značilnosti, njegove prednosti in slabosti ter osnovno terminologijo. V nadaljevanju naj predstavi obstoječe storitve računalništva v oblaku ter s praktičnim primerom pokaže možno uporabo. V okviru diplomske naloge naj kandidat preuči tudi ključne izzive na tem področju in možen razvoj dogodkov v prihodnosti.

Mentor:
prof. dr. Marko Bajec

Dekan:
prof. dr. Franc Solina

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01559/2009
Datum: 05.04.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JAKOB MERLJAK**

Naslov: **RAČUNALNIŠTVO V OBLAKU**
CLOUD COMPUTING

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Tako imenovano »računalništvo v oblaku« je vedno bolj priljubljeno področje računalništva. Osnovna ideja, ki se skriva za omenjenim pojmom, je ponuditi visoko zmogljive, zanesljive, prilagodljive in cenovno ugodne računalniške storitve, ki bodo uporabnikom dostopne preko enostavnih spletnih vmesnikov oziroma prek spleta.

Kandidat naj v svojem diplomskem delu predstavi omenjeno področje, našteje zgodovinske dogodke, ki so pripeljali do razvoja omenjenega pristopa, opiše ključne značilnosti, njegove prednosti in slabosti ter osnovno terminologijo. V nadaljevanju naj predstavi obstoječe storitve računalništva v oblaku ter s praktičnim primerom pokaže možno uporabo. V okviru diplomske naloge naj kandidat preuči tudi ključne izzive na tem področju in možen razvoj dogodkov v prihodnosti.

Mentor:
prof. dr. Marko Bajec

Dekan:
prof. dr. Franc Solina

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani **Jakob Merljak**,

z vpisno številko **63040100**,

sem avtor diplomskega dela z naslovom:

Računalništvo v oblaku.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno, pod mentorstvom **izr. prof. dr. Marka Bajca**;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 21.9.2009 Podpis avtorja:

Zahvala

Iskrena zahvala gre prijateljem in sorodnikom za vso podporo v času pisanja diplomske naloge. Zahvaljujem se doc. dr. Boštjanu Slivniku za pomoč pri praktičnem delu naloge, mentorju izr. prof. dr. Marku Bajcu za strokovni pregled diplomske naloge, Ani Merljak Bratuš za lektoriranje ter Nadji Berlot Dukšič za prevod povzetka v angleščino.

Kazalo

Povzetek **1**

1 Uvod **5**

2 Računalništvo v oblaku **6**
2.1 Zgodovina 7
2.2 Neenotnost definicij 8
2.3 Značilnosti 9
2.4 Kaj računalništvo v oblaku ni 10
2.5 Vloge in komponente 10
2.6 Prednosti 11
2.7 Slabosti 12
2.8 Alternative 13
2.9 Oblak ali oblaki? 14

3 Ponudba storitev **15**
3.1 Programska oprema 15
3.2 Platforme 23
3.3 Shranjevanje podatkov 25
3.4 Infrastruktura 26
3.5 Druge storitve 29

4 Primer uporabe: prevajalnik v oblaku **30**
4.1 Prevajalnik 31
4.2 Implementacija prevajalnika 32
4.3 Prilagoditev za izvajanje v oblaku 45
4.4 Težave in rešitve 53
4.5 Možnosti izboljšav 54
4.6 Nadzorna plošča, verzije, opravila 54

5 Sklep **56**
5.1 Izzivi 56
5.2 Prihodnost 57

Dodatki **59**

A Specifikacija jezika miniPascal **59**
A.1 Leksikalna pravila 59
A.2 Sintaksna pravila 60
A.3 Semantična pravila 63

Seznam slik **66**
Seznam izvorne kode **67**

Literatura **68**

Seznam uporabljenih kratic in simbolov

AJAX - Asynchronous JavaScript and XML
API - Application Programming Interface
ASP - Application Service Provider
ATM - Asynchronous Transfer Mode
AWS - Amazon Web Services
CRM - Customer Relationship Management
CSS - Cascading Style Sheets
EC2 - Amazon Elastic Computer Cloud
ERP - Enterprise Resource Planning
GAE - Google App Engine
GQL - Google App Engine Query Language
HRM - Human Resource Management
IaaS - Infrastructure as a Service
IDE - Integrated Development Environment
IP - Internet Protocol
IT - Information Technology
JSON - JavaScript Object Notation
JSP - Java Server Pages
PaaS - Platform as a Service
PC - Personal Computer
PDF - Portable Document Format
RAID - Redundant Array of Inexpensive Disks
REST - Representational State Transfer
S3 - Amazon Simple Storage Service
SaaS - Software as a Service
SCM - Supply Chain Management
SLA - Service Level Agreement
SVN - Subversion Version Control System

QoS - Quality of Service
VLAN - Virtual Local Area Network
XMPP - Extensible Messaging and Presence Protocol
XML - Extensible Markup Language

Povzetek

Diplomska naloga obravnava trenutno vročo temo računalništva - področje računalništva v oblaku.

Po kratkem uvodu avtor v drugem poglavju predstavi glavne značilnosti računalništva v oblaku, obenem pa ugotavlja, da natančna in splošno sprejeta definicija računalništva v oblaku ne obstaja. Navedeni so nekateri zgodovinski dogodki, ki so pripeljali do današnjega stanja, predstavljena je terminologija, prednosti in slabosti računalništva v oblaku kot tudi možne alternative.

Tretje poglavje prinaša pregled sedaj obstoječe ponudbe storitev računalništva v oblaku. Pregled je smiselno razdeljen na aplikacije, platforme, infrastrukturo, hrambo podatkov ter druge storitev.

V četrtem poglavju avtor s praktičnim primerom predstavi uporabo koncepta računalništva v oblaku. Praktični del diplomske naloge zajema implementacijo prevajalnika iz programskega jezika miniPascal v zbirni jezik arhitekture MMIX in prilagoditev prevajalnika za izvajanje na platformi Google App Engine.

Zaključno poglavje predstavi nekatere dosedanje izkušnje pri uporabi računalništva v oblaku, izpostavi izzive nadaljnjega razvoja področja in poda napoved, da se področju obeta še svetla prihodnost.

Ključne besede:

računalništvo v oblaku, programska oprema kot storitev, platforma kot storitev, infrastruktura kot storitev, spletne storitve

Abstract

The thesis deals with a currently very popular theme - cloud computing.

A short introduction is followed by presentation of the cloud computing characteristics and the finding that there is no accurate and generally accepted definition of this term. The author presents some historical events that led to the current situation, terminology, pros and cons of cloud computing and its alternatives.

Third chapter features the overview of already available cloud computing services, divided into applications, platforms, infrastructure, data storage and other kinds of services.

In the fourth chapter the author presents the use of cloud computing concept on a practical example. This practical part includes implementation of compiler from the miniPascal programming language to the MMIX assembly language and compiler's adjustment for running on the Google App Engine platform.

In the last chapter, some experiences of using cloud computing are presented, challenges of further development in this field are exposed and the prediction is made that cloud computing has a great future ahead.

Key words:

cloud computing, software as a service, platform as a service, infrastructure as a service, web services

Poglavje 1

Uvod

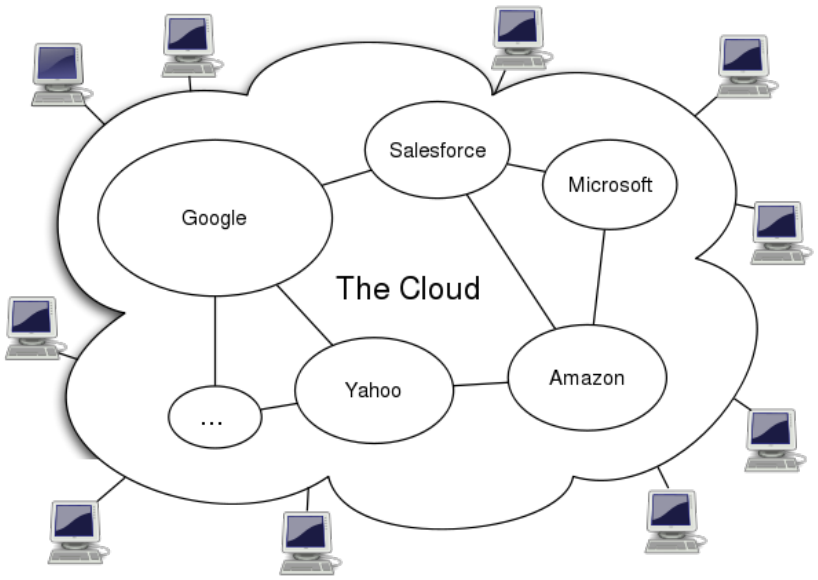
Zadnje čase vroča tema računalništva je tako imenovano računalništvo v oblaku. Pod tem imenom se skriva vrsta tehnologij in principov, ki lahko dodobra spremenijo podobo računalništva, kot ga poznamo danes, pa tudi vrsta storitev, ki jih že sedaj redno uporabljamo.

Kaj se točno skriva za tem pojmom, kaj je že na voljo in kaj lahko še pričakujemo, je računalništvo v oblaku že dovolj zrelo, zanesljivo in varno ter ali je to res prihodnost računalništva, so vprašanja, ki so poleg radovednosti botrovala mojemu izboru diplomske teme. V tej diplomski nalogi bom poskušal odgovoriti na zgoraj postavljena vprašanja in predstaviti praktični primer uporabe.

Poglavje 2

Računalništvo v oblaku

Izraz *računalništvo v oblaku* se je začel široko uveljavljati šele v zadnjih letih, izvira pa iz shematske skice svetovnega spleta (glej sliko 2.1), ki ga poenostavljeno predstavljamo z oblakom, na katerega in preko katerega so povezane računalniške naprave: strežniki, osebni računalniki, prenosni računalniki, lahki odjemalci, mobilni telefoni, zadnja leta pa tudi čedalje več drugih elektronskih naprav, ki s pridom uporabljajo svetovni splet v takšne in drugačne namene.



Slika 2.1: Računalništvo v oblaku je ime dobilo po poenostavljeni shemi svetovnega spleta.

V tej poenostavljeni shemi so vse podrobnosti - tudi zapletena sestava omrežij svetovnega spleta - skrite. Podrobnosti nas namreč velikokrat sploh ne zanimajo, ampak želimo le izkoristiti vse možnosti, ki nam jih svetovni splet ponuja. **Glavna ideja računalništva v oblaku je: uporabnikom ponuditi visoko zmogljive, zanesljive, prilagodljive in cenovno ugodne računalniške storitve, dostopne preko enostavnih spletnih vmesnikov**. Pri tem so podrobnosti očem skrite, vemo le, da se stvari izvajajo ***nekje v oblaku***.

2.1 Zgodovina

V šestdesetih letih 20. stoletja je John McCarthy napovedal [9], da bo računanje (uporaba računalniških zmogljivosti) kmalu na voljo kot javna storitev, a je podobo računalništva dodobra spremenil IBM s predstavitvijo osebnega računalnika (PC). Minilo je kar nekaj deset let, da se je ideja ponovno prijela. Ključno vlogo pri tem sedaj igra internet.

Običajna strežba spletnih strani se je z leti vse od pojava svetovnega spleta dalje vedno bolj razvijala, s pomočjo skriptnih strežniških jezikov so strani postajale vedno bolj bogate, dinamične. Pojavil se je pojem *web 2.0*, ki naj bi označeval drugo generacijo spleta, ki ni več statičen in omogoča enostavno izmenjavo informacij, sodelovanje in prispevanje k vsebini (na primer: družabna spletišča, blogi, *wiki*-ji, strani za objavo in izmenjavo fotografij, glasbe in video posnetkov). Sočasno se pojem **programska oprema kot storitev** (*software as a service*) začne čedalje bolj uporabljati za označevanje spletnih aplikacij.

Uporaba le-teh se je dejansko že zelo približala uporabi namiznih, krajevno nameščenih programov, s to razliko, da se tako aplikacija kot podatki nahajajo pri ponudniku storitve. Do teh aplikacij dostopamo preko spletnega brskalnika, ki postaja čedalje bolj univerzalno in nepogrešljivo orodje. Vsega tega smo se že dodobra navadili in se skoraj ne zavedamo več razlike med namiznimi in spletnimi aplikacijami. Verjetno najbolj znan in široko uporabljen primer spletnih aplikacij so zmogljivi spletni odjemalci za elektronsko pošto ter spletni pisarniški programi (urejevalniki besedila in preglednic). Vse te storitve so v prvi meri namenjene končnim uporabnikom, čeprav so ponekod na voljo tudi programski vmesniki (API), namenjeni razvijalcem - programerjem.

Korak naprej je pomenila predstavitev storitev prednostno namenjenih razvijalcem. To sta koncepta **platforma kot storitev** (*platform as a service*) in **infrastruktura kot storitev** (*infrastructure as a service*). Pri prvem gre za to, da programerju zagotovimo platformo, na katero lahko postavi svoje

aplikacije, razvite v enem izmed podprtih programskih jezikov. Platforma mora seveda poskrbeti za osnovne zahteve, ki jih aplikacija sicer pričakuje od operacijskega sistema (in/ali virtualnega stroja, pri interpretiranih jezikih). Nekomu, ki želi razviti in na svetovni splet postaviti neko lastno aplikacijo, torej ni potrebno več postavljati lastnih strežnikov, s čimer se izogne tudi zahtevnim nastavitvam sistema, zagotavljanju fizične in omrežne varnosti ter vzdrževanju sistema. Cilj infrastrukture kot storitve pa je zagotoviti najem (virtualizirane in dinamično razširljive) računalniške infrastrukture, na katero stranke shranjujejo svoje podatke ali postavijo svoje lastne celotne sisteme, ohranijo nadzor nad programsko opremo, znebijo pa se skrbi z načrtovanjem, postavljanjem in vzdrževanjem strojne opreme, skrbi za varnostno kopiranje podatkov ...

Med prvimi so sicer idejo računalniških storitev skušali uresničiti ponudniki ASP (*application service provider*). Izraz oblak ima izvor v telefoniji, komercialna uporaba imena *cloud* (oblak) pa se v začetku devetdesetih let nanaša na velika ATM omrežja. Ob prelomu tisočletja se izraz **računalništvo v oblaku** že širše uporablja, tedaj predvsem za SaaS. Pomembno vlogo pri razvoju in popularizaciji koncepta odigrata Salesforce.com (1999) in Amazon Web Services (2005). Do leta 2007 je bil pojem že široko sprejet in v zadnjem času vedno bolj pridobiva na popularnosti.

2.2 Neenotnost definicij

Natančne in splošno sprejete definicije računalništva v oblaku ni. Če bi povprašali dvajset strokovnjakov o definiciji, bi zelo verjetno dobili prav toliko različnih odgovorov.

Trenutna definicija na Wikipediji se glasi: "Računalništvo v oblaku je paradigma računalništva, pri kateri so dinamično razširljiva in pogosto virtualizirana (računalniška) sredstva na voljo kot storitev preko interneta. Uporabniki ne potrebujejo poznavanja in strokovnega znanja o tehnologiji ali nadzora nad infrastrukturo v *oblaku*, ki storitve omogoča."[9]

Druga definicija pravi: "Uporaba internetnih strežnikov v računske namene. Ponudniki računalništva v oblaku postavijo podatkovne centre, dostopne preko interneta, ter uporabnikom tržijo procesorski čas."[13]

Nekoliko oprijemljivejša je definicija: "Samopostrežne spletne aplikacije, platforme, storitve in infrastruktura, na zahtevo, obračunane po principu *plačaj po porabi*, dostopne s spletnim brskalnikom, aplikacijo ali preko programskega vmesnika."[11, 12]

Spet nekoliko drugačna je sledeča definicija: "Računalništvo v oblaku je realizacija razvoja in uporabe računalniške tehnologije, temelječe na internetu in dostavljene s strani ekosistema ponudnikov."[4]

Po besedah Simona Wardleya, pa pojav računalništva v oblaku pomeni prehod od produktov k storitvam in je na področje bolje gledati s stališča prakse: "Računalništvo v oblaku je splošen pojem, uporabljen za opis prelomne transformacije IT v smeri storitveno usmerjene ekonomije, kot posledica ekonomskih, kulturnih in tehnoloških razmer."[8]

Prave in edine veljavne definicije torej ni mogoče postaviti. Lahko pa izpostavimo nekatere pomembne, splošno sprejete značilnosti.

2.3 Značilnosti

V splošnem koncept računalništva v oblaku združuje vse tri pojme: programska oprema kot storitev, platforma kot storitev ter infrastruktura kot storitev (kot tudi nekaj drugih storitev in tehnologij). Vsem trem konceptom je skupno, da računalniške zmogljivosti zagotavlja ponudnik storitev, uporabnik pa potrebuje zgolj osnovno zmogljiv računalnik ter do storitev dostopa preko omrežja. Ključne značilnosti so [9]:

- **agilnost** (uporabnik si lahko hitro in poceni priskrbi potrebno tehnologijo),
- v splošnem **nižji stroški** (odsotnost večine investicijskih stroškov, nižji prag za vstop na trg, plačilo po dejanski porabi sredstvi),
- **neodvisnost od lokacije in naprave** (do storitev lahko dostopamo od skoraj kjerkoli, prav tako izvajanje ni nujno vezano na točno določen strežnik),
- ***multi tenancy*** (iste instance strežnikov, platform in aplikacij strežejo množici uporabnikov),
- **zanesljivost** (redundanca strežnikov in podatkov, varnostni mehanizmi),
- **razširljivost** (glede na potrebe, v skoraj realnemu času),
- **varnost** (izboljšana zaradi naprednejših in celovitejših sistemov za zagotavljanje varnosti, ki si jih manjša podjetja ne morejo privoščiti, a vnaša skrbi zaradi izgube nadzora nad občutljivimi podatki) in
- **boljša izraba sredstev**.

2.4 Kaj računalništvo v oblaku ni

Pojem računalništva v oblaku se pogosto zamenjuje z mrežnim računalništvom (*grid computing* - oblika porazdeljenega računalništva, kjer za obdelavo zelo zahtevnih opravil uporabimo navidezni super računalnik, sestavljen iz gruče ohlapno povezanih omreženih računalnikov, ki sodelujejo za dosego skupnega cilja), *uslužnostnim* računalništvom (*utility computing* - ponujanje računalniških virov, kot so računska zmogljivost in hranjenje podatkov, kot merjeno storitev) ter avtonomnim računalništvom (sistemi, sposobni samo-upravljanja). Računalništvo v oblaku vsekakor temelji na omenjenih konceptih, a jih poizkuša še nadgraditi.

2.5 Vloge in komponente

- **Ponudnik** (*cloud computing provider*) - oseba ali podjetje, ki zagotavlja in trži storitve računalništva v oblaku. Običajno so potrebni veliki vložki ter veliko znanja za vzpostavitev in upravljanje takega računalniškega sistema.
- **Uporabnik** (*cloud computing user*) - oseba ali podjetje, ki pri ponudniku najame/zakupi storitve računalništva v oblaku. Potrebno znanje in vložki so minimalni.
- **Odjemalec** (*cloud client*) - strojna in/ali programska oprema, ki za delovanje izkorišča storitve računalništva v oblaku, je posebej namenjena uporabi teh storitev oziroma je od teh storitev življenjsko odvisna. Primeri: spletni brskalniki, lahki odjemalci, mobilne platforme.
- **Storitve** (*cloud services*) - storitve dostavljene in porabljene v realnem času preko spleta, namenjene tako končnim uporabnikom kot tudi povezavi z drugimi komponentami računalništva v oblaku. Primeri: sporočilne vrste, plačilni sistemi, iskanje, identifikacija in drugo.
- **Aplikacija** (*cloud application*) - program ali aplikacija, na voljo kot storitev, običajno brez potrebe po nameščanju in poganjanju na lokalnem računalniku.
- **Platforma** (*cloud platform*) - platforma/programski sklad, na voljo kot storitev, ki omogoča uporabniku postavitev (*deploy*) lastnih aplikacij, brez stroškov in dela povezanega z nakupom in upravljanjem strojne in programske opreme na kateri aplikacija temelji.

- **Hranjenje podatkov** (*cloud storage*) - storitev hrambe podatkov v oblaku.
- **Infrastruktura** (*cloud infrastructure*) - tipično virtualizirana infrastruktura (strežniki), na voljo kot storitev.

Slika 2.2: Plasti računalništva v oblaku

2.6 Prednosti

Že na prvi pogled je prednosti res veliko. Z najemom storitev računalništva v oblaku se najprej izognemo postavljanju lastne infrastrukture (investicijski stroški) ter s tem povezanega načrtovanja in predvidevanja potrebnih zmogljivosti. Potrebujemo le osnovno strojno in programsko opremo ter dostop do svetovnega spleta. To zagotavlja nizke vstopne stroške, že takoj na začetku pa nam je na voljo širok nabor rešitev, zelo zmogljivi sistemi in najnovejša tehnologija.

Ker nam ponudniki zagotavljajo dinamično razširljivost, se ni bati, da bi bili žrtev prevelikega uspeha lastne dejavnosti in ne bi uspeli slediti naraščajočim zahtevam po računalniških zmogljivostih. Prav tako se izognemo primeru, ko bi precenili svoj potencial in kupili preveč zmogljivo (in primerno drago) strojno in programsko opremo. Z računalništvom v oblaku najamemo in plačamo le tisto, kar potrebujemo in uporabljamo - večinoma po primerljivih ali ugodnejših cenah. Razloge za nižje cene lahko najdemo v ekonomiji obsega ter v dejstvu, da deljenje (pogosto virtualiziranih) ponudnikovih računalniških zmogljivosti med mnogimi uporabniki prispeva k boljši izrabi sredstev (manj časa so nedejavne - *idle*). Obenem lahko v primeru, da neke storitve ne želimo več uporabljati, vsak trenutek prekinemo pogodbo ter se izognemo nadaljnjim stroškom.

Dostop do zakupljenih storitev imamo tako rekoč kdajkoli in od kjerkoli (kjer je dostop do interneta), tehnologija nam je na voljo ravno pravočasno (*just-in-time*). Ponudniki storitev zagotavljajo praktično neprekinjeno delovanje in dosegljivost sistema. Zaradi centralizacije je nadzor nad podatki olajšan, izguba službenega prenosnega računalnika pa ne pomeni nujno tragedije, saj so lahko vsi podatki varno shranjeni v oblaku, na prenosniku pa nameščen zgolj odjemalec.

Nenazadnje, v podjetju potrebujemo manj IT strokovnjakov za upravljanje z računalniškimi sistemi, saj za večino nalog poskrbijo ponudniki storitev. Upravljanje vključuje redne posodobitve strojne in programske opreme, fizično in omrežno varovanje, optimizacijo, lahko tudi varnostno kopiranje podatkov in drugo. Zaposleni v podjetju se zato lahko osredotočijo na strateške tehnologije, ne na podporne.

2.7 Slabosti

Da ni vse zlato kar se sveti, govorijo sledeči pomisleki glede uporabe računalništva v oblaku. V primeru, da bi investicijski stroški predstavljali le majhen delež celotnih stroškov, so začetni prihranki majhni, kasneje pa operativni stroški lahko celo večji. Ob tem imamo občutno zmanjšan nadzor nad strojno opremo.

Varnostni pristopi so kljub zagotovilom velikokrat pomanjkljivi, segajo pa od preprostih prijav zgolj z geslom, odsotnosti šifriranja, do generiranja prešibkih oziroma ponavljajočih se šifrirnih ključev zaradi pomanjkanja zadostne entropije sistema (veliko število strežniških instanc na isti strojni opremi). Veriga pa je tako močna, kot njen najšibkejši del.

Ob oblici (osebnih) podatkov, ki se dnevno zbirajo, obdelujejo in shranjujejo, je še pomembnejše vprašanje zaupanja - ali lahko ponudniku zaupamo, da bo spoštoval zasebnost in v podatke ne bo posegal ali jih celo zlorabil? Je sistem dovolj dobro zastavljen, da ne bo prišlo do razkritja podatkov drugim uporabnikom? Marsikatero podjetje v nobenem primeru ne želi izgubiti nadzora nad podatki. Občasno tudi ni povsem jasno, kdo je lastnik podatkov, shranjenih v oblaku - uporabnik ali ponudnik. Pravna zaščita uporabnikov je zaradi globalnega trga neenotna.

Z uporabo storitev v oblaku se sicer res lahko izognemo nameščanju programske opreme, je pa uvedba kompleksnih sistemov še vedno zelo zahtevna. Problematičen je lahko tudi prenos podatkov in nastavitev iz, na primer, obstoječih poslovnih rešitev, ki jih neko podjetje uporablja.

Zaradi pomanjkanja standardov obstaja bojazen nezdružljivosti med ponudniki ter posledično odvisnost od ponudnika, ki smo si ga izbrali. Pri tem smo omejeni na storitve, ki jih ponuja, ter takšne, kot jih ponuja. Svoboda in ustvarjalnost pri razvoju sta omejeni. Na tem mestu se zdi utemeljeno tudi vprašanje, kaj bi se zgodilo, če bi ponudnik propadel in prenehal tržiti svoje storitve. Lahko celoten sistem in vsi naši podatki čez noč izginejo? Nenazadnje so marsikatere storitve še nezrele in v razvoju.

2.8 Alternative

Kot alternativa se je pojavil pojem **zasebni oblak** (*private cloud*), ki označuje emulacijo računalništva v oblaku znotraj zasebnih omrežij. Običajno se dosega z virtualizacijo in avtomatizacijo. Tako naj bi pridobili nekatere prednosti računalništva v oblaku (prilagodljivost, boljši izkoristek infrastrukture) in se izognili pastem klasičnega računalništva v oblaku - t.i. javnega oblaka (*public cloud*). Poudarja se predvsem nadzor nad podatki in sploh celotnim sistemom, ter neodvisnost od ponudnikov. Kritike takih rešitev letijo predvsem na dejstvo, da zasebni oblaki hkrati prinesejo investicijske stroške in problem načrtovanja, postavljanja in vzdrževanja računalniškega sistema, kar izniči nekaj bistvenih prednosti, zaradi česar je računalništvo v oblaku privlačno. Obenem plačujemo tudi za neizkoriščene računalniške zmogljivosti.

Za uporabo javnega oblaka je tipično, da zagotavlja hitrejšo in cenejšo pot do postavitve zahtevanega računalniškega sistema, upravljanje z njimi pa je poenostavljeno. Na drugi strani zasebni oblaki zagotavljajo neposreden nadzor nad podatki, varnostjo in kvaliteto storitve, lažje naj bi jih bilo tudi vključiti v obstoječo IT infrastrukturo. Zdi se verjetno, da se na dolgi rok lahko izkažejo za cenejšo rešitev. [10]

O prihodnosti zasebnih oblakov se pojavljajo mešane napovedi. Manjša in srednje velika podjetja si - v nasprotju z velikimi korporacijami, kjer zasebnim oblakom napovedujejo svetlo prihodnost - najverjetnejše ne bodo mogla privoščiti zasebnih oblakov in bodo večino računalniških potreb zadovoljila s storitvami javnega oblaka. Po nekaterih napovedih pa naj bi bili zasebni oblaki samo vmesni korak do računalništva v javnem (pravem, internetnem) oblaku.

Orodja, ki nam omogočajo, da na lokalni računalniški gruči ustvarimo svoj zasebni oblak, so na primer odprtokodni projekt Eucalyptus, 3Tera AppLogic, Joyent Cloud Control, IBM CloudBurst ter Oracle Platform for SaaS.

O **hibridnem oblaku** (*hybrid cloud*) govorimo, ko kombiniramo uporabo zasebnih z javnim oblakom.

2.9 Oblak ali oblaki?

Na eni strani imamo internetni oblak, na drugi so se pojavili zasebni oblaki. Je torej pravilneje pisati: računalništvo v oblakih? Uporabnik namreč razlike mogoče sploh ne opazi.

Za lažjo odločitev si še enkrat poglejmo ključne značilnosti (razdelek 2.3). Skupne točke obeh svetov so *multy tenancy*, zanesljivost ob ustreznih varnostnih mehanizmih, prilagodljivost ter boljša izraba sredstev. Uporaba (javnega) oblaka prinaša agilnost in nizke vstopne stroške. Postavljanje lastne infrastrukture in zasebnega oblaka pa nikakor ni poceni, še manj hitro opravilo. Neodvisnost od lokacije in naprave lahko dosežemo tudi z zasebnimi oblaki, a so ponavadi namenjeni uporabi znotraj lokalnih omrežij. Razširljivost tudi ni nekaj kar si lahko privoščimo čez noč, saj je ob (pričakovanem) stalnem povečevanju potreb, potrebno sistem stalno nadgrajevati. Tako pa smo spet pri investicijskih stroških in zahtevnem vzdrževanju.

Med javnim oblakom in zasebnimi oblaki obstaja torej kar nekaj bistvenih razlik. V pričujoči diplomski nalogi se bom osredotočil na javno dostopne storitve, zato bom dosledno uporabljal izraz računalništvo v oblaku.

Poglavje 3

Ponudba storitev

Pri pregledu ponudbe storitev računalništva v oblaku sem najprej načrtoval popis ponudnikov, ki javno razglašajo, da ponujajo računalniške storitve v oblaku, ter opis njihovih storitev. Vendar se je izkazalo, da je računalništvo v oblaku širši pojem in vključuje tudi množico storitev, ki se morda sploh ne oglašujejo kot storitve računalništva v oblaku. Obenem se je število ponudnikov nepregledno povečalo. Zato sem raje naredil pregled po posameznih področjih uporabe. Pregled seveda ni popoln, saj je ponudba res široka, obenem pa vsak dan nastajajo nove storitve.

3.1 Programska oprema

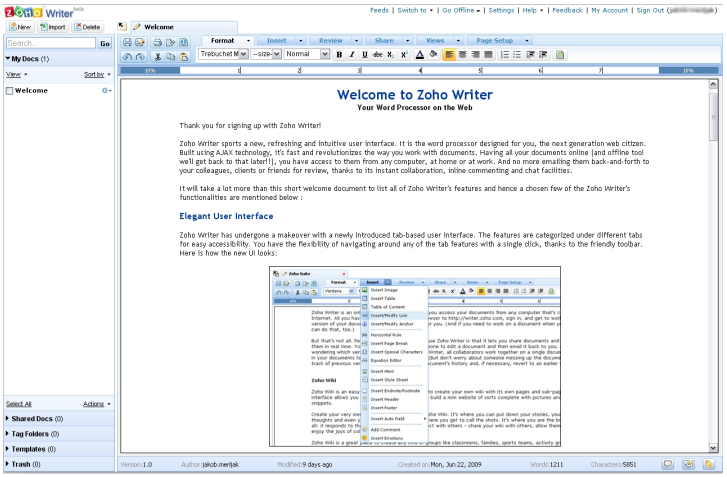
V kategoriji programske opreme kot storitve najdemo res obilico storitev. Pravzaprav je težko narediti črto in ločiti bogate spletne strani od tistih strani, ki ponujajo programsko opremo kot storitev. Poskusil sem predstaviti zgolj storitve, ki za uporabnika delujejo kot programi/aplikacije in po možnosti za podlago in shranjevanje podatkov uporabljajo principe računalništva v oblaku. Na ta način sem se izognil mešanju v zgodbo spletne trgovine, forume, bloge, socialne mreže ...

Pisarniški programi

S splošno uveljavitvijo elektronske pošte kot pomembnega načina za dogovarjanje med ljudmi, tako v zasebnem kot poslovnem okolju, se je pojavila želja po dostopu do sporočil od kjerkoli in to ne samo do novejših, ampak tudi do arhiva prejetih in poslanih sporočil. Rešitev so predstavile zmogljive

spletne aplikacije za elektronsko pošto (Google Gmail, Microsoft Hotmail, Yahoo! Mail ...). Ponudniki spletno pošto shranjujejo (v oblaku), kar zagotavlja dostop od kjerkoli, indeksirajo jo za učinkovitejše iskanje po sporočilih, razvrščajo v pogovore, omogočajo označevanje, filtriranje ... V marsičem so tovrstne spletne aplikacije že povsem enakovredne namiznim odjemalcem za elektronsko pošto, po nekaterih zmogljivostih pa jih celo prekašajo (možnost neposrednega sporočanja, zmogljivo iskanje, uporabniški dodatki, povezovanje z drugimi spletnimi storitvami ...). Predvsem pa nam je elektronska pošta na voljo od kjerkoli, poskrbljeno je za varnostno kopiranje podatkov.

V splet se čedalje bolj selijo tudi klasični pisarniški programi - urejevalniki besedila, preglednic, predstavitev - ter aplikacije za upravljanje dokumentov in skupinsko delo (Google Docs, Microsoft Office Live, OpenGoo, Oracle Beehive, ThinkFree Office, Zimbra Collaboration Suite, Zoho Office Suite ...). Spletne pisarne se sicer po zmogljivosti in naboru funkcij ne morejo kosati z izredno zmogljivimi namiznimi paketi (Microsoft Office, Open Office, Star Office), a znano je, da večina uporabnikov uporablja le osnovne funkcije pisarniških programov. Nasprotno pa uporabniki cenijo dostop od kjerkoli ter enostavno deljenje datotek med uporabniki in skupinsko delo, ki jim omogočajo spletni pisarniški programi ... Dobrodošla je tudi lastnost sledenja spremembam in revizija različic dokumentov. S stalnim dopolnjevanjem se tovrstnim storitvam kaže svetla prihodnost.



Slika 3.1: Zoho Writer

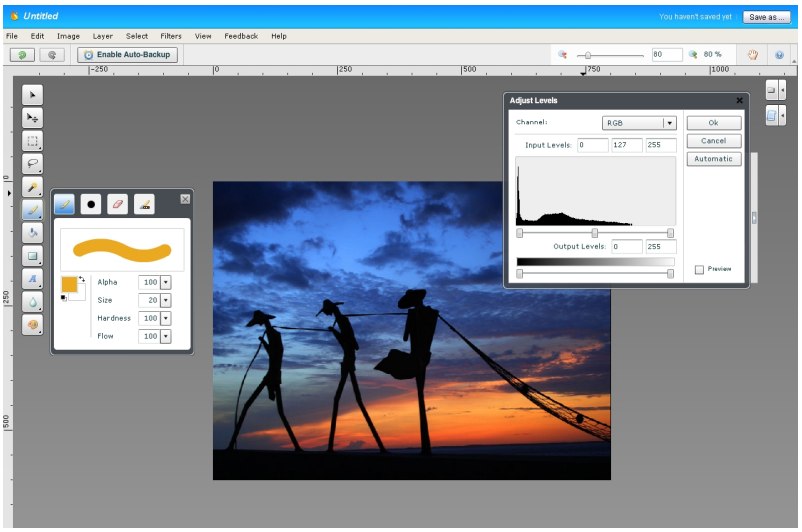
Nepogrešljivi so tudi koledarji, ki se prav tako selijo v oblak. Poleg očitnega dostopa od kjerkoli, ponujajo še obveščanje po elektronski pošti ter celo s SMS obvestili. Pomemben del je zmožnost sinhronizacije z različnimi napravami.

Zelo uporabna je možnost uporabe lastne spletne domene (recimo v paketu Google Apps), kjer pridobimo vse prednosti omenjenih spletnih storitev, hkrati pa uporabljamo svojo domeno, brez zahtevnega in zamudnega konfiguriranja lastnega spletnega, poštnega in/ali aplikacijskega strežnika.

Večpredstavnost

Med bolj priljubljenimi so storitve shranjevanja/objavljanja fotografij (Flickr, Picasa Web Albums, ImageShack, Panoramio, Photobucket), avdio (Imeem, Exaudioshare) ter video posnetkov (YouTube, Vimeo, blip.tv, Dailymotion, Yahoo! Video, MySpaceTV, MSN Video). Samo shranjevanje in objavljanje pa ne bi bilo nič posebnega in bi ga najbrž zajeli v sklopu shranjevanja podatkov v oblaku, če ne bi bilo dodatnih, posebej večpredstavnostnim datotekam namenjenih storitev.

Ponudniki shranjevanja fotografij največkrat omogočajo še vsaj avtomatsko izdelavo fotogalerij, prikaz kot predstavitev, dodajanje podatkov o lokaciji posnete fotografije, prikaz na zemljevidu ... Fotografije pa lahko v oblaku ne le shranjujemo, ampak tudi obdelujemo. To omogočajo storitve Picnik, Adobe Photoshop Express, Aviary Phoenix, s katerimi lahko slike osnovno obdelujemo z najpogostejšimi orodji. Aviary v svoji ponudbi ponuja še urejevalnik vektorskih slik Raven ter nekaj preprostejših orodij, na primer Falcon za hiter zajem in označevanje slik ter Toucan za izdelavo barvnih palet.

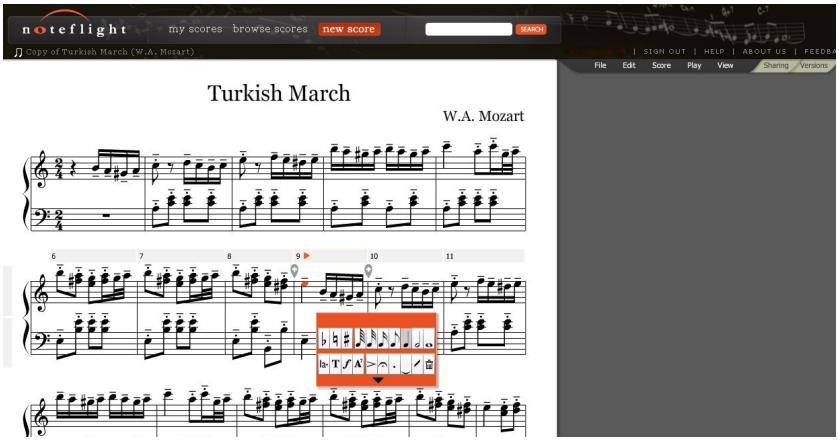


Slika 3.2: Aviary Phoenix - urejanje fotografij v oblaku

Če se je včasih obdelava zvoka in videa preko interneta zdela znanstvena fantastika, pa se vrata čedalje bolj odpirajo tudi v to smer. Med prvimi na področju spletnega urejanja avdio posnetkov se je pojavil DigiMix. Storitev je pred časom kupil Aviary in trenutno ni dosegljiva. Doslej poznana razvojna različica pa je že omogočala večstezno urejanje zvoka v realnem času. Podobna in delujoča storitev je Jamglue (za delovanje uporablja Amazonovo infrastrukturo). Na video področju veliko obeta storitev Jaycut. Nabor funkcij je sicer še dokaj skromen, a preprost video posnetek (z osnovnimi prehodi in učinki) že lahko zmontiramo izključno z uporabo te spletne aplikacije.

Glasba in govor

Tudi za glasbenike se najde rešitev. Noteflight je spletna aplikacija za ustvarjanje, urejanje in izmenjavo notnih zapisov. Omogoča vse, kar (nezahtevni) uporabniki pričakujejo: ustvarjanje novih dokumentov, vnašanje not in besedila, transponiranje, predvajanje (z zvoki najpogostejših inštrumentov) ...



Slika 3.3: Noteflight - spletni urejevalnik notnih zapisov

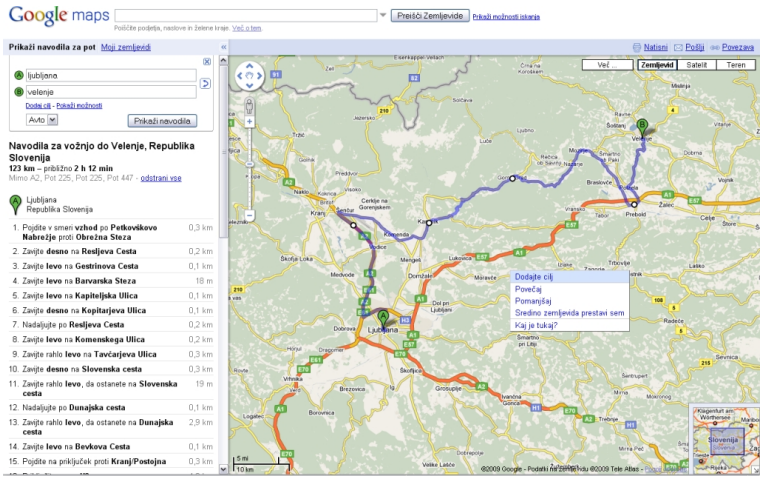
Bistvena razlika z namiznimi programi je zmožnost izmenjave datotek med uporabniki, sodelovanje pri ustvarjanju ter enostavna vgradnja na spletne strani. Glasbeno notacijo lahko tako enostavno vgradimo na blog, kjer je na ogled vsem obiskovalcem, brez potrebe po nalaganju vtičnikov, vgrajeni flash predvajalnik pa omogoča celo predvajanje glasbe. Storitev za delovanje izkorišča Amazonove storitve S3 in EC2.

Običajnemu tipkanju iskanega izraza pri spletnem iskanju, s katerim se srečujemo vsak dan in je na majhnih napravah (kot so mobilni telefoni) lahko tudi zamudno, pa se lahko tudi izognemo. Google ponuja storitev, s pomočjo katere iskani izraz enostavno izgovorimo, posnetek se nato prenese preko omrežja, kjer se v oblaku izvede prepoznavanje govora in običajna poizvedba, rezultati pa so nato posredovani nazaj na telefon.

Zemljevidi

Pri shranjevanju fotografij je bilo omenjeno označevanje položaja posnetka na zemljevidu. Običajno lahko položaj slik prikažemo na zemljevidu ponudnika storitve shranjevanja fotografij ali pa na zemljevidih enega izmed ponudnikov spletnih zemljevidov.

Spletni zemljevidi so v zadnjih letih močno napredovali, postali vedno bolj natančni in omogočajo vedno več. Poleg osnovnega iskanja krajev, prikaza zemljevida in satelitskih posnetkov, storitve omogočajo tudi izračun poti (celo z dinamičnim prilagajanjem poti, tako da pot enostavno "povlečemo"!), izpis navodil za potovanje, prikaz znamenitosti, 3D pogled na nekatera večja mesta in drugo. Prednost spletnih različic je tudi v stalnem posodabljanju, uporaba storitev pa je večinoma brezplačna. Najbolj znani ponudniki so Google Maps, Live Search Maps, Yahoo! Maps, MapQuest ter ViaMichelin.



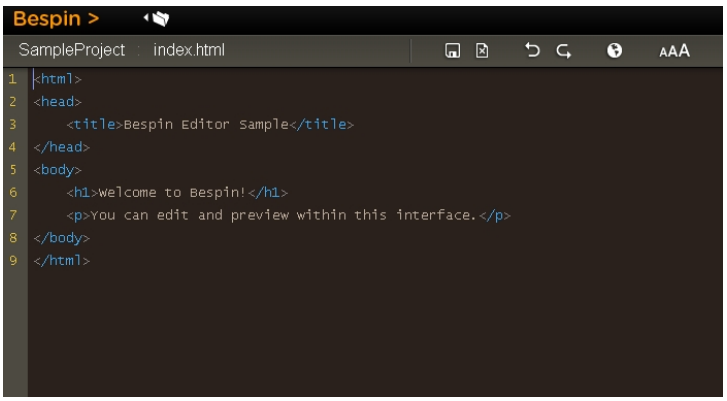
Slika 3.4: Načrtovanje poti z Google Maps

Prevajanje

Oblak nam pomaga tudi, ko potrebujemo hiter prevod besedila v/iz tujega jezika. Izbiramo lahko med prevajalniki: Yahoo! Bebelfish, Google Translator, WorldLingo, Promt, Windows Live Translator, ImTranslator in drugimi, čeprav nekateri v resnici ne prevajajo sami, ampak uporabljajo storitve drugih (primer: Nice Translator). Omeniti velja, da so prevodi zgolj osnovni (nekateri temeljijo tudi na statistiki) in niso (še) primerni za profesionalno prevajanje v tuj jezik.

Programiranje

V splet se selijo tudi urejevalniki programske kode. V organizaciji Mozilla, najbolj znani po spletnem brskalniku Firefox, razvijajo platformo Bespin, katere namen je predstaviti spletni urejevalnik kode, ki bi omogočal vse, kar omogočajo običajni urejevalniki kode, obenem pa bi deloval v oblaku in bil tako dostopen vedno in od kjerkoli. Trenutno je podprto označevanje sintakse jezikov HTML, CSS, PHP, Arduino in JavaScript.

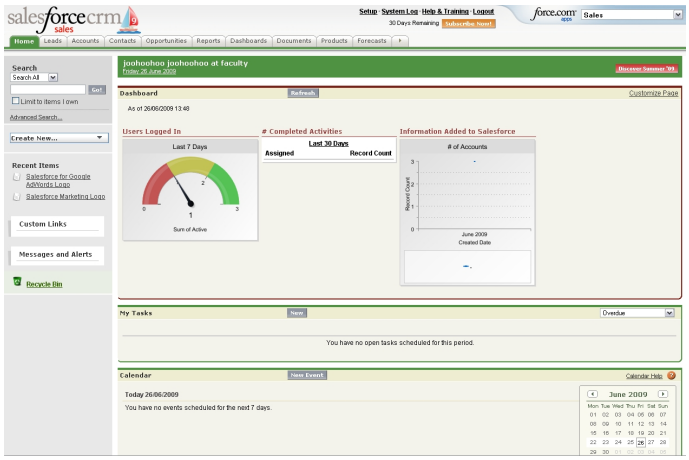


Slika 3.5: Bespin - spletni urejevalnik kode

Zanimiva spletna aplikacija prihaja iz podjetja Sun. Zembly, kot je storitev poimenovana, je spletno razvojno programsko okolje (IDE), namenjeno programiranju aplikacij za uporabo v socialnih mrežah, kot so Facebook, Twitter, Meebo. Posebnost storitve *zembly* je, da kombinira lastnosti klasičnih razvojnih okolij (na primer urejevalnik programske kode) s pristopi, znanimi iz socialnih mrež in wiki strani, kar odpira možnost novega pristopa k razvoju aplikacij. Uporabniki storitve (ki je tudi sama neke vrste socialna mreža) lahko med sabo komunicirajo in sodelujejo pri razvoju aplikacij.

Poslovne aplikacije

Podjetjem so namenjene aplikacije upravljanja odnosov s strankami (CRM, na primer Oracle CRM, Salesforce.com, SugarCRM, Morph eXchange CRM), upravljanja oskrbovalne verige (SCM), celovite poslovne aplikacije (ERP, na primer Compiere Cloud Edition), aplikacije upravljanja človeških virov (HRM, na primer EmplX HRMS), računovodske (Xero, Clarity Accounting, Quick Books Online) in druge poslovne aplikacije (Oracle Sourcing, SAP E-Sourcing, Morph eXchange ECM). Nekateri ponudniki ponujajo posamezne specializirane aplikacije, drugi zaokrožene zbirke (na primer Netsuite in Paglo).



Slika 3.6: Salesforce.com CRM

Varnost

Na tem mestu je prav, da omenimo tudi prvo predstavljeno protivirusno aplikacijo, ki izkorišča koncept računalništva v oblaku. Panda Cloud Antivirus omogoča omejeno varovanje računalnika in je namenjen predvsem manj zmogljivim računalnikom. Aplikacija deluje tako, da sumljive datoteke pošilja v zmogljive strežnike ponudnika protivirusne rešitve, kjer se "v oblaku" datoteke pregledajo za morebitnimi virusi.

Spletno storitev odkrivanja zlonamerne programske kode (*malware*) je pred kratkim predstavilo podjetje SurfRight. Hitman Pro, kot je ta storitev poimenovana, za odkrivanje škodljivih programov uporablja vedenjsko preiskovanje (*behavioural scan*). Tudi tukaj potrebujemo lokalno nameščen program, ki zbira podatke o sumljivih aplikacijah in podatke pošilja v preverjanje na strežnike ponudnika storitve.

Ob obilici storitev, ki jih uporabljamo, se rado zgodi, da se množica uporabniških imen in gesel neobvladljivo povečuje. V takih primerih nam pomagajo upravitelji gesel. Na voljo je kar nekaj spletnih različic tovrstnih aplikacij: Passpack, Clipperz, Mashed Life, Online Password Manager, Xecrets ... Večina jih omogoča varno hranjenje gesel ter avtomatsko prijavo na spletne strani. Pred uporabo storitev se splača preveriti, če so naša gesla na strežniku shranjena šifrirano, v obliki, ki je niti ponudnik storitve ne more prebrati brez naše vednosti.

Igre

Ne glede na starost se ljudje vedno radi igramo. Na splet je bilo prenesenih že veliko število iger, večinoma preko tehnologije Flash, a prava revolucija šele prihaja. OnLive je spletna storitev (trenutno še v razvoju), ki naj bi združila svetova iger in računalništva v oblaku. Celotno procesiranje, sinhronizacija in renderiranje slike se izvaja na strežnikih ponudnika, preko omrežja pa se prenašajo le slika ter ukazi iz igralne konzole. Za igranje najzmogljivejših iger naj bi torej kmalu zadoščal nezmogljiv računalnik (dovolj, da zna predvajati video in ima hitro internetno povezavo) oz. set-top box s povezavo na televizor.

Drugo

Zanimive so storitve pretvorbe datotek. Movavi Online Video Converter ter Mediaconverter.com omogočata pretvorbo med različnimi formati video in avdio posnetkov, ki so bodisi objavljeni na spletu bodisi se nahajajo na našem računalniku. Podjetje Adobe omogoča storitev Create PDF, s pomočjo katere lahko naše dokumente in slike pretvorimo v zelo razširjen format PDF. Korak dlje gresta storitvi Free File Convert ter Zamzar, ki združujeta oboje in omogočata pretvorbo med različnimi formati dokumentov, slik, avdio in video posnetkov ter celo stisnjenih arhivov.

Ob koncu velja omeniti še RightScale, ki ponuja spletno aplikacijo za upravljanje z zakupljenimi storitvami infrastrukture v oblaku za različne ponudnike IaaS (*infrastructure as a service*).

Kot je razvidno iz pregleda, je ponudba res velika in široka. Iz dneva v dan nastajajo nove storitve in čedalje manj je programov, ki bi obstajali le v namizni različici in ne bi imeli vsaj približek v spletni storitvi tega ali onega ponudnika. V prihodnje si lahko obetamo izboljšave in razširitev obstoječih storitev kot tudi obilico novih aplikacij.

3.2 Platforme

Pregled nadaljujemo pri platformah, ki omogočajo enostaven razvoj in postavitev aplikacij v oblaku. Vsem platformam je skupna značilnost, da uporabniku ni potrebno skrbeti za strojno opremo, operacijski sistem, varnostne kopije in optimizacijo, varnost ali posodobitve platforme. Rešitve segajo od enostavnih platform za posamezen programski jezik ali nekaj izbranih programskih jezikov, preko rešitev, ki bi jih lahko opisali kot operacijski sistem v oblaku, do specializiranih lastnih rešitev (kjer pravzaprav ni pomembno kateri programski jeziki poganjajo aplikacije). Običajno je del storitve tudi takšna ali drugačna podatkovna baza.

Google App Engine je preprosta platforma, ki gosti naše aplikacije, napisane v enem izmed podprtih programskih jezikov. Zaenkrat sta (z nekaterimi omejitvami) podprta jezika Python in Java. Na voljo so programski vmesniki (API), preko katerih lahko dostopamo do nekaterih zmogljivosti platforme. Uporabna je možnost prijav s pomočjo razširjenih uporabniških Google računov, uporaba enostavne podatkovne baze BigTable s pomočjo poizvedovalnega jezika GQL, nastavljanje načrtovanih opravil ... Število strežnikov, na katerih teče aplikacija, se dinamično spreminja glede na zahteve. Za lažji nadzor nad aplikacijami imamo na voljo nadzorno ploščo, kjer si lahko pregledamo sistemski dnevnik (log), preverimo koliko prostora in procesorskega časa porablja naša aplikacija, k sodelovanju povabimo druge razvijalce ... Osnovna uporaba (do 10 aplikacij, omejena uporaba prostora, procesorskega časa, zahtevkov ...) je brezplačna, zahtevnejša plačljiva. Ena večja pomanjkljivosti platforme App Engine je trenutna odsotnost storitve shranjevanja večjih količin podatkov. V prihodnje naj bi pri Googlu to pomanjkljivost odpravili.

Na področje računalništva v oblaku se je odločil vstopiti tudi Microsoft. Trenutno so njihove storitve v stanju tehnološkega predogleda (*technology preview).* **Windows Azure Services** je skupek storitev namenjen razvoju, gostovanju in upravljanju spletnih storitev v Microsofotovem oblaku. V središču je *oblačni operacijski sistem* Windows Azure. Pri Microsoftu zagotavljajo, da bo na platformi mogoče poganjati tako aplikacije in spletne strani napisane v okolju .NET, kot tudi v nekaterih drugih razširjenih programskih/skriptnih jezikih, recimo v PHP. Dodatne storitve so: SQL Services (shranjevanja strukturiranih, delno strukturiranih in nestukturiranih podatkov), .NET Services (zagotavljanje ključnih gradnikov za tvorbo aplikacij namenjenih gostovanju na platformi Azure), Live Services (skupek gradnikov za upravljanje z uporabniškimi podatki; omogočali naj bi razvoj socialnih aplikacij, sinhronizacijo podatkov med različnimi napravami) ter druge storitve.

Eden izmed prvih ponudnikov platforme v oblaku je bilo podjetje Heroku z istoimensko platformo. Platforma **Heroku** omogoča gostovanje aplikacij, napisanih v okolju Ruby on Rails. V pomoč je podatkovna baza MySQL. Podobno storitev ponuja **Engine Yard**, medtem ko **Morph eXchange Platform** podpira nekoliko več programskih jezikov (Ruby on Rails, Java , Grails, PHP). **Joyent Smart Platform** je namenjena razvoju spletnih aplikacij s pomočjo jezikov HTML, JavaScript in Git. **Rackspace Cloud Sites** je nekakšen hibrid med Linux in Windows strežniki in omogoča istočasno izvajanje aplikacij, napisanih za Windows ali Linux

Precej uspešna je platforma **Force.com**, na kateri temelji prav tako uspešna storitev Salesforce.com (opisana pri poslovnih SaaS aplikacijah). Platforma omogoča gostovanje spletnih strani in spletnih aplikacij, tudi takih, ki jih razvijalci naprej tržijo kot SaaS. Bistvena prednost, zaradi katere so jo uporabniki sprejeli, pa so zmogljiva orodja za razvoj aplikacij, s katerimi lahko aplikacijo precej hitreje postavimo. Vse potrebno postorimo kar iz spletnega brskalnika.

Razvoju in postavitvi poslovnih aplikacij je namenjena tudi platforma **Bungee Connect**. Če se želimo izogniti programiranju, vseeno pa bi si želeli kompleksno spletno aplikacijo, rešitve ponujajo **WorkXpress**, **Wolf Frameworks** ter **Rollbase**. Aplikacijo gradimo kar v spletnem brskalniku s klikanjem: izbiramo vnaprej pripravljene gradnike, določamo njihove lastnosti, izgled, akcije. Mimogrede, orodja s katerimi gradimo naše aplikacije, bi lahko uvrstili med SaaS rešitve.

Svojevrstna platforma je **Facebook Platform**, namenjena razvoju in gostovanju aplikacij, ki se vključujejo v izredno priljubljeno socialno mrežo Facebook.

Omenimo še nekaj specializiranih platform, ki so namenjena razvoju aplikacij, katerih bistveni sestavni del je komunikacija (govor in telefonija). Primer takih platform so: **Tropo**, **Twilio** in **IfByPhone**.

3.3 Shranjevanje podatkov

Najprej bi lahko med ponudnike shranjevanja podatkov uvrstili storitve shranjevanja naših večpredstavnostnih datotek v oblaku (Flickr, blip.tv, Imeem, Panoramio, Photobucket, Picasa Web Albums, Vimeo, YouTube in drugi - glej pregled ponudbe SaaS, razdelek 3.2) ter aplikacije za izmenjavo datotek in skupinsko delo (Google Docs, Microsoft Office Live, Oracle Beehive, Zimbra Collaboration Suite ... - glej razdelek 3.2). Tudi druge rešitve seveda shranjujejo naše podatke v oblaku, v naslednjih odstavkih pa se bom osredotočil na specializirane rešitve.

Na tržišču je že nekaj časa precej ponudnikov (tudi slovenskih), ki omogočajo shranjevanje varnostnih kopij naših podatkov (poljubnih datotek) na oddaljeni lokaciji. Najpreprostejše so rešitve, ki delujejo kot oddaljen diskovni pogon. Naše podatke enostavno prepišemo na navidezni disk oziroma povlečemo v posebno mapo, nakar se prenesejo na strežnike ponudnika. Očitni slabosti takega načina sta naslednji: če podatke kopiramo, se lahko pojavi neusklajenost (na primer: na strežniku ni najnovejša različica datotek), če pa oddaljeno shrambo uporabljamo za primarno shrambo, smo omejeni s hitrostjo omrežne povezave, v primeru izpada povezave pa so datoteke nedosegljive. Naprednejše storitve omogočajo (preko dodatnih lokalno nameščenih programov): avtomatsko izdelavo varnostnih kopij, sledenje spremembam datotek in sproten prenos na strežnik, *prirastno* posodabljanje varnostnih kopij (*incremental backup* - prenesejo se le spremenjeni deli datotek) ter stiskanje in šifriranje podatkov. Nekateri izmed ponudnikov storitev so: Backblaze, Backup City, Carbonite, Elephant Drive, IBackup, Jungle Disk, Memopal, Mozy, NomaDesk, SkyDrive.

Vse bolj pa se shranjevanje v oblaku uveljavlja v pravem pomenu besede - ne le kot varnostna kopija, ampak kot primarna shramba ter kot storitev za druge storitve računalništva v oblaku. V ta namen potrebujemo programske vmesnike (API). Primer takih ponudnikov so Amazon S3, Box.net, Digital Bucket, Nirvanix Storage Delivery Network, Rackspace Cloud Files, Sun Cloud Storage Service, Windows Azure Storage.

Tipično je, da uporabniki od ponudnika pričakujemo, da se podatki samodejno varnostno kopirajo in po možnosti porazdelijo na več strežnikov, kar zagotavlja visoko razpoložljivost in zanesljivost hrambe.

3.4 Infrastruktura

Najprej nekaj lastnosti, ki so skupne vsem opisanim ponudnikom in jih ne bom posebej našteval: na voljo so nam *on-demand* navidezni strežniki v več zmogljivostnih razredih, izognemo se investicijskim vložkom v infrastrukturo, do delujočega strežnika nas loči le nekaj minut, stroški pa se obračunajo glede na dejansko porabo sistemskih sredstev. Uporabnik je običajno omejen le z zmogljivostmi strežnikov in izbiro platforme, nadalje ima proste roke. Avtomatsko je poskrbljeno za redundanco ter porazdelitev, po potrebi lahko sistem enostavno razširimo (dodamo strežniške instance, povečamo zakupljen prostor ali zmogljivosti navideznega strežnika). Zaradi odsotnosti dolgoročnih pogodb, načeloma nismo vezani na enega ponudnika, saj lahko kadarkoli prekinemo pogodbo. Pri menjavi ponudnika pa lahko pride do problemov z združljivostjo.

V podjetju Amazon so pred leti izvedli notranjo reorganizacijo računalniške infrastrukture. Pri tem pa uporabili koncepte, ki so danes temelj za računalništvo v oblaku. Rezultati so bili zelo dobri, zaradi pozitivnih izkušenj uporabe znotraj podjetja, pa so se odločili del svojih zmogljivosti ponuditi tudi javnosti. Pod imenom **Amazon Web Services** (AWS) že od leta 2002 tržijo svojo ponudbo storitev računalništva v oblaku. Amazon Elastic Compute Cloud (EC2) je spletna storitev, ki zagotavlja razširljivo računalniško infrastrukturo v oblaku. Amazon s pomočjo virtualizacijskega okolja, ki temelji na odprtokodni tehnologiji Xen, omogoča gostovanje virtualnih strežnikov. Poganjamo lahko katerega izmed vnaprej pripravljenih navideznih strojev ali pa izdelamo svojega in ga shranimo na S3. Podprti so različni operacijski sistemi. Na voljo so nam različno zmogljivi sistemi, prednost storitve pa je hitra prilagodljivost, saj lahko v času nekaj minut enostavno povečamo ali zmanjšamo število delujočih primerkov (instanc) strežnikov, glede na naše dejanske potrebe. Strežnike med drugim lahko nadziramo tudi s pomočjo dodatka za spletni brskalnik, Amazon pa zagotavlja dobro povezanost z ostalimi storitvami AWS ter zanesljivost in varnost storitve. Dogovorjen nivo storitve (SLA) za EC2 znaša 99,95%. Amazon je pred kratkim predstavil še storitev imenovano Virtual Public Cloud. Strežniki v tem navideznem zasebnem oblaku so dostopni samo preko povezave VPN, kar naj bi zagotovilo varno in povsem transparentno vključitev navideznega zasebnega oblaka v obstoječe računalniške sisteme.

Uporaba storitve EC2 zahteva nekaj pazljivosti pri delu s podatki. Če pride do napake na strežniku in ponovnega zagona instance, izgubimo vse naše podatke na navideznem disku. Pri ponovnem zagonu se namreč sistem vzpostavi na novo iz prvotne slike sistema. Če želimo trajnost podatkov, moramo uporabiti storitev Amazon Elastic Block Storage ali Amazon S3. Alternativa

je uporaba storitve Amazon SimpleDB, ki zagotavlja osnovno funkcionalnost podatkovne baze. Storitev je tesno povezana z EC2 in S3, ki skupaj zagotavljajo shranjevanje, procesiranje in poizvedovanje po naboru podatkov v oblaku. Za razliko od tradicionalnih podatkovnih baz, je uporaba SimpleDB zelo poenostavljena, saj ne zahteva sheme niti modeliranja podatkovne baze niti vzdrževanja indeksiranja in piljenja zmogljivosti (*performance tuning*).

S podjetjem Amazon je pogodbo o sodelovanju podpisalo veliko število partnerjev, ki preko EC2 ponujajo svoje predpripravljene sisteme. Primer so številni operacijski sistemi (Microsoft Windows Server, Open Solaris, več Linux distribucij), podatkovne baze (IBM DB2, MySQL, Oracle 11g, Microsoft SQL Server), spletni strežniki (Apache, IIS, IBM Lotus Web Content Management, IBM WebSphere Portal Server), aplikacijski strežniki (IBM WebSphere ApplicationServer, Java Application Server, Oracle WebLogic) ter vrsta drugih rešitev.

Podjetje **GoGrid** v svoji ponudbi ponuja gostovanje predpripravljenih in uporabniških virtualiziranih sistemov, shranjevanje podatkov, API vmesnike in storitev Cloud Connect za povezovanje GoGrid strežniške infrastrukture s privatnimi oblaki v hibridni oblak. Obenem se v boj na tržišču spušča z nekaterimi ugodnostmi (promocije, uporabniška podpora, novejše verzije strežniških platform).

Virtualizirane strežnike ponuja tudi Rackspace s storitvijo **Rackspace Cloud Servers**. Že na spletni strani dajo vedeti, da konkurirajo predvsem podjetju Amazon. V primerjavi z Amazonovo storitvijo EC2, ponujajo nekoliko več izbire pri izbiri zmogljivostni instanc, predvsem pri skromnejših strežnikih za tiste, ki ne potrebujejo visokih zmogljivosti. Pomembnejša je razlika v zagotavljanju trajnosti podatkov. Če pri EC2 potrebujemo dodatne storitve shranjevanja, pri Rackspace Cloud Servers že v osnovi zagotavljajo trajnost podatkov na diskih (*persistent storage*), kljub morebitni odpovedi strežnika. Nekoliko je poenostavljeno tudi delo z IP naslovi, saj imajo navidezni strežniki fiksen privaten IP naslov, ki se ohrani tudi po ponovnem zagonu. Zakupljene instance imajo zagotovljeno minimalno zmogljivost CPU, obenem pa lahko neomejeno izkoriščajo proste procesorske zmogljivosti. Strežnike lahko upravljamo s pomočjo spletne nadzorne plošče, v pomoč so nam še dodatna orodja. Rackspace ne omogoča poganjanje strežnikov Windows Server, programski vmesniki (API) pa so še v razvojni fazi. Oglaševani dogovorjeni nivo storitve (SLA) je 100%.

Podobno storitev predstavlja **FlexiScale**. Tako kot pri Rackspace, se storitvi pozna, da je nastajala nekoliko kasneje kot Amazonova ponudba ter z neposrednim namenom zagotavljanja infrastrukture v oblaku, zato omogoča

nekatere novejše in preprostejše pristope. Tako je poskrbljeno za trajnost podatkov, poenostavljeno je delo z IP naslovi, velikost strežnikov (pomnilnik in disk) je prilagodljivejša, poleg zagotovljene lahko strežniki izkoriščajo tudi ostale proste procesorske zmogljivosti. Posebnost je omogočanje virtualnega lokalnega omrežja (VLAN) med uporabnikovimi strežniškimi instancami. Oglaševani SLA je 100%.

Povsem nasprotno ponudnik **Slicehost** ne obljublja dogovorjenega nivoja storitve, ker naj bi bil ta podatek pri tekmecih zavajajoč. Zagotavljajo le, da se bodo potrudili po najboljših močeh, da bo sistem deloval zanesljivo in brez prekinitev. V ponudbi je na voljo več različno zmogljivih instanc strežnikov, prostor na trdih diskih v polju RAID10, avtomatizirana izdelava varnostnih kopij, aplikacija za dostop z iPhone in telefoni na platformi Android, poleg zagotovljene zmogljivosti lahko strežniki izkoriščajo tudi proste procesorske zmogljivosti. Izbiramo lahko med različnimi Linux sistemi.

Svojo rešitev je predstavilo tudi podjetje Sun, ki ima sicer že dolgoletne izkušnje s strežniki in virtualizacijo. **Sun Open Cloud** temelji na strežnikih Open Solaris ter tehnologijah Sun Grid Engine in Java. Načrtovanje arhitekture računalniškega sistema postorimo kar v spletnem brskalniku, s klikanjem, vlečenjem in povezovanjem pripravljenih omrežnih elementov (usmerjevalniki, spletni, podatkovni in aplikacijski strežniki).

Podjetje **Joyent** ponuja storitev **Public Cloud**, ki prav tako temelji na strežnikih Open Solaris, pa tudi drugače se pohvalijo z uporabo odprtih standardov, kar naj bi bilo zagotovilo za prenosljivost in združljivost z drugimi sistemi. Obenem je poskrbljeno za trajnost podatkov, omogočeni so statični IP naslovi in dovoljeno je izkoriščanti proste procesorske zmogljivosti.

Konkurenčne prednosti IaaS ponudnika **AppNexus** naj bi bile preglednost (*transparency*), posluh do želja uporabnikov, napredni varnostni mehanizmi, zasebno VLAN omrežje, 24 urna telefonska podpora ter 100% SLA.

IBM je pred kratkim napovedal ponudbo gostovanja delovnih okolij v oblaku. Podjetja, ki se bodo za to odločila, bodo za zaposlene morala priskrbeti le osnovno strojno opremo (osnovno zmogljiv osebni računalnik ali lahki odjemalec z dostopom do svetovnega spleta), vsi podatki in navidezni stroji pa se bodo nahajali v IBM-ovih podatkovnih centrih.

Drugi ponudniki IaaS so še: Tsunamic Technologies, Iland, Skytap.

3.5 Druge storitve

Amazon Elastic MapReduce je storitev namenjena raziskovalcem, analitikom podatkov in drugim, ki potrebujejo učinkovito procesiranje velikanskih količin podatkov (na primer: spletno indeksiranje, podatkovno rudarjenje, analiza dnevniških datotek, strojno učenje, finančne analize, znanstvene simulacije in raziskave). Temelji na odprtokodni platformi Apache Hadoop in izkorišča infrastrukturo EC2 (procesiranje) in S3 (vir vhodnih podatkov in shranjevanje rezultatov), pri čemer se uporabniku ni potrebno ukvarjati z zahtevnim postavljanjem, upravljanjem in uglaševanjem Hadoop računalniške gruče. Platforma Hadoop razdeljuje podatke na manjše dele, ki se lahko obdelujejo vzporedno ter končno združi obdelane podatke v končni rezultat. Amazon obljublja, da je storitev prilagodljiva, enostavna za uporabo, zanesljiva, varna in poceni.

Storitev Amazon Mechanical Turk aplikacijam omogoča koordinacijo s človeškimi viri za opravila, ki so odvisna od posredovanja človeške inteligence (*human intelligence tasks*).

S storitvijo Simple Queue Service (SQS) podjetje Amazon ponuja storitev sporočilne vrste za shranjevanje sporočil, ki se prenašajo med računalniki. S tem lahko preprečimo izgubo informacij med prenosom in se izognemo zahtevi, da so naše komponente vedno dosegljive. Dostop je standardiziran in ne zahteva posebne programske opreme.

Storitev Amazon CloudFront je namenjena enostavni dostavi vsebin preko spleta. Našim podatkom, ki so shranjeni na S3, določimo unikatno ime distribucije in jo javno objavimo. Zahtevki po prenosu so avtomatsko preusmerjeni na najbližjo lokacijo, kjer so shranjeni naši podatki, kar zagotavlja kar najmanjše zakasnitve in kar najvišje prenosne hitrosti.

Med storitve računalništva v oblaku pa bi lahko prišteli tudi storitve prilagojenega spletnega iskanja (Alexa, Google Custom Search, Yahoo! BOSS), plačilne storitve (Amazon Flexible Payments Service, Google Checkout, Pay Pal), storitve enotne identifikacije uporabnika (OAuth, OpenID) ter druge.

Poglavje 4

Primer uporabe: prevajalnik v oblaku

Kot praktični primer uporabe storitev računalništva v oblaku sem se odločil izdelati spletno aplikacijo ter jo postaviti na eno izmed obstoječih platform (PaaS - *platform as a service*).

Za začetek sem pregledal ponudbo in preveril, kateri ponudniki ponujajo brezplačno ali časovno/funkcijsko omejeno uporabo. V ožji izbor sem uvrstil Salesforce.com, Wolf Frameworks ter Google App Engine. Po kakšnem tednu preizkušanja je ostal v igri samo en ponudnik. Odločujoč faktor pri izbiri je na koncu predstavljalo razvojno okolje. Salesforce omogoča razvoj v spletnem brskalniku, žal pa je uporabniški vmesnik precej zasičen z raznimi funkcijami, zato se brez natančne vizije, kaj želimo, hitro izgubimo. Ponudba Wolf Framework je izgledala nekoliko bolj obetavno s svojo enostavno shemo, a se je izkazalo, da stvari ne delujejo tako, kot bi morale. Povrhu je kar nekaj funkcij delovalo le v brskalniku Internet Explorer, kar je za današnje razmere nesprejemljivo. Pri Google App Engine tovrstnih problemov ni, saj se razvijalec pri razvoju aplikacij lahko zanese na lokalno nameščena in preizkušena razvojna okolja (IDE), nato pa projekt le prenese na strežnike.

Google App Engine je bil torej logična izbira, vseeno pa brez težav ni šlo. Platformo sem najprej hotel uporabiti v navezavi s programskim jezikom Python, a se je izkazalo, da je omejitev samo na uporabo modulov, ki so napisani v *čistem* Pythonu, prevelika ovira. Večina modulov za Python je namreč delno napisana v programskem jeziku C, ki drastično pohitri nekatere operacije (recimo obdelavo slik). Zaradi te omejitve nisem mogel uporabiti na naši fakulteti razvitega paketa za strojno učenje Orange, prav tako se je izjalovil poskus s knjižnico Python Imaging Library (PIL).

Končna odločitev: navezava platforme Google App Engine ter programskega jezika Java. V tem primeru lahko uporabimo katerekoli knjižnice in module, ki so prevedeni v Java *bytecode*.

4.1 Prevajalnik

Prevajalnik je računalniški program, ki prevaja izvorno programsko kodo, napisano v nekem (višje nivojskem) računalniškem jeziku, v drug računalniški jezik. Navadno je rezultat prevajanja programska koda v nižje nivojskem programskem jeziku, vmesni kodi (pri interpretiranih jezikih) ali strojnem jeziku. Prevajanje se izvaja z namenom izdelave izvršljivih programov/strojne kode.

Prevajalnik lahko razdelimo na prednji iz zadnji del. Prednji del enostavnejšega prevajalnika sestavljajo leksikalna analiza, sintaksna analiza, semantična analiza in prevajanje v vmesno kodo, zadnji del pa optimizacija vmesne kode, generiranje ciljne kode ter optimizacija le-te. Pogosto prevajalnik vsebuje še več faz. Prednji del je napisan glede na izvorni programski jezik, zadnji del pa za ciljno platformo, na kateri se bo aplikacija izvajala. [1, 2, 9]

Nekateri programski jeziki, na primer zelo razširjeni C, potrebujejo posebno fazo imenovano **predprocesiranje** (*preprocessing*), ki podpira zamenjavo makro ukazov ter pogojno prevajanje.

V drugem (oziroma prvem) koraku prevajanja se izvede **leksikalna analiza** izvorne programske kode. Izvorno kodo programa razbijemo na posamezne besede, preverimo, če vsebuje dovoljene znake, izločimo komentarje, ter vsaki besedi določimo vrsto (rezervirana beseda ali znak, ime ...), njeno znakovno predstavitev in položaj v izvorni kodi. Pri tem vedno sprejmemo najdaljši možni del vhoda.

Namen **sintaksne analize** je preveriti sintakso izvorne kode (vrstni red osnovnih simbolov), ki mora ustrezati sintaksnim pravilom jezika; ter zgraditi sintaksno drevo, namenjeno nadaljnjim stopnjam prevajanja. Sintakso jezika lahko opišemo z (nedvoumno) kontekstno neodvisno gramatiko.

Sledi **semantična analiza**, ki sintaksnemu drevesu doda semantično informacijo ter izdela simbolno tabelo (preslikava med imeni in njihovim pomenom). Pri tem se običajno opravi analiza in preverjanje tipov spremenljivk in izrazov. Prav tako je lahko zahtevano, da so vse lokalne spremenljivke inicializirane pred uporabo.

Če se predhodne stopnje izvedejo brez napak, se izvede **prevajanje v vmesno kodo** (interna predstavitev izvorne kode), prevajalnik pa preide v zadnji del. Zadnji del prevajalnika, za razliko od prednjega, ni več odvisen od

značilnosti vhodnega programskega jezika, pač pa je vezan na ciljno platformo oziroma arhitekturo.

Analiza vmesne kode je proces zbiranja informacij o vmesni kodi. Primeri: analiza toka podatkov, analiza odvisnosti, analiza kazalcev in sopomenk itd. Natančna analiza je osnova za pomembno stopnjo **optimizacije vmesne kode**, katere namen je pohitritev izvajanja. To dosežemo na različne načine. Priljubljeni pristopi optimizacije so odstranjevanje mrtve (nedosegljive) kode, *inline* vključevanje funkcij in procedur (izognemo se klicanju in vračanju iz funkcij in procedur), optimizacija programskih zank, optimizirano dodeljevanje registrov, propagiranje konstant ...

Zaključni korak prevajanja je **generiranje ciljne kode** - pretvorba optimizirane vmesne kode v ciljni računalniški jezik.

4.2 Implementacija prevajalnika

Izdelava prevajalnika je zahteven in dolgotrajen projekt, ki vključuje tako zahtevno implementacijo kot tudi stalno preverjanje pravilnosti delovanja, izboljševanje algoritmov optimizacije kode, prilagajanje spremembam v specifikaciji izvornega jezika in spremembam ciljne platforme/arhitekture. Vzrok nepravilnega delovanja programov zaradi napake v prevajalniku je namreč izredno težko odkriti.

Zahtevnega dela smo se oprijeli pri predmetu Prevajalniki, v 4.letniku univerzitetnega študija Računalništva in informatike, na smeri Programska oprema. Ker je en semester, kolikor je trajalo izvajanje predmeta, občutno premalo za izdelavo prevajalnika, je bilo potrebno sprejeti nekaj kompromisov. Za izvorni jezik smo izbrali poenostavljeno različico programskega jezika Pascal, poimenovano miniPascal (specifikacija jezika je v prilogi A). Optimizaciji vmesne kode smo se skoraj v celoti izognili. Ciljna podlaga izdelanega prevajalnika je arhitektura MMIX [5], prevajalnik pa je skoraj v celoti napisan v programskem jeziku Java, izjema so leksikalna in sintaksna pravila.

Leksikalni analizator

Leksikalni analizator bi lahko v celoti razvili s svojo logiko, vendar to vodi do zelo kompleksne in nepregledne izvorne kode prevajalnika. Namesto tega se poslužujemo orodij, ki na osnovi regularnih izrazov, s katerimi opišemo leksikalna pravila jezika, zgenerirajo leksikalni analizator, ki ga potem vključimo kot del prevajalnika. Za programski jezik Java je v ta namen na voljo knjižnica JFlex (*The Fast Scanner Generator for Java*).

Datoteka `.lex`, na podlagi katere JFlex zgradi leksikalni analizator (oziroma *scanner*), ima posebno strukturo, vsebuje pa mešanico regularnih izrazov ter Java programsko kodo, s katero opišemo, kako naj analizator obravnava simbole in besede izvorne datoteke. Na delu kode vidimo regularne izraze, ki opisujejo imena, znakovne, logične in številčne konstante, prazne znake ter del seznama rezerviranih besed:

```
%{
 int comment_depth = 0;

 private Symbol sym(int type){
  return new Symbol(type, yyline+1, yycolumn+1, yytext());
 }
%}

IDENTIFIER      = [A-Za-z_][A-Za-z0-9_]*
STRING_CONST    = \'([^'\t\n]|\'\')*\'
BOOLEAN_CONST   = true|false
INTEGER_CONST   = [0-9]+

[ \t\n\r]+      { }

<YYINITIAL> {
 "and"      { return sym(AND); }
 "array"    { return sym(ARRAY); }
 "begin"    { return sym(BEGIN); }
 "const"    { return sym(CONST); }
 "div"      { return sym(DIV); }
 "do"       { return sym(DO); }
 "else"     { return sym(ELSE); }
 ...
```

Izvorna koda 4.1: pascal.lex

V primeru najdenih napak v izvorni kodi, se le-te izpišejo, izvajanje pa se kljub temu nadaljuje (če je preostali del izvorne kode v redu, se zgolj izpusti nelegalne znake in nadaljuje s prevajanjem). Zgrajen leksikalni analizator (v našem primeru `PascalLex.java`) je sicer namenjen za uporabo skupaj s sintaksnim analizatorjem in ga samostojno ponavadi ne uporabljamo.

Sintaksni analizator

Tudi pri tem koraku si pomagamo z namenskimi orodji. Sintakso jezika naprej opišemo z nedvoumno kontekstno neodvisno gramatiko, nato pa s pomočjo orodij, kot je Java Cup (*Parser Generator for Java*), zgeneriramo sintaksni

analizator in ga skupaj z leksikalnim analizatorjem vključimo kot del prevajalnika.

Del gramatike, ki opisuje sintakso jezika miniPascal, prikazuje izvorna koda 4.2. Vidimo, da je program sestavljen iz glave programa, deklaracij konstant, tipov, spremenljivk in podprogramov ter sestavljenega stavka, ki mu sledi pika. Glavo programa napoveduje rezervirana beseda `program`, ki ji sledi ime programa in podpičje. If stavek pa določajo rezervirane besede `if`, `then` in (neobvezno) `else`:

```
program
 ::= program_heading:i const_decl_opt:c type_decl_opt:t
        var_decl_opt:v sub_decl_opt:sub cmpnd_stmt:cs DOT
   {:  RESULT = new SynProgram(i,c,t,v,sub,cs); :}
 ;

program_heading
 ::= PROGRAM identifier:i SEMIC
   {:  RESULT = i; :}
 ;

if_stmt
 ::= IF:i exp:e THEN stmt:s1
   {: RESULT = new SynIfStmt(ileft,iright,e,s1,null); :}
  | IF:i exp:e THEN stmt:s1 ELSE stmt:s2
   {: RESULT = new SynIfStmt(ileft,iright,e,s1,s2); :}
 ;
```

Izvorna koda 4.2: pascal.cup

Prevajanje izvorne datoteke *source.pas* sprožimo z naslednjim nizom ukazov, pri čemer je `PascalSyn.java` izgrajen sintaksni analizator:

```
try {
 FileReader src = new FileReader("source.pas");
 PascalLex scanner = new PascalLex(src);
 PascalSyn parser = new PascalSyn(scanner);
 SynTree synTree = (SynProgram)(parser.parse().value);
}
catch (Exception ex) {}
```

Izvorna koda 4.3: Main.java (leksikalna in sintaksna analiza)

Iz zgornje kode je razvidno, da ob uspešni leksikalni in sintaksni analizi (t.j. v primeru, ko ni najdenih napak) prevajalnik zgradi sintaksno drevo, namenjeno nadaljnjem prevajanju. Če so odkrite napake v sintaksi izvorne kode, se le-te izpišejo, prevajanje pa se prekine. Prav tako se izpišejo druga

opozorila (na primer inicializacija spremenljivke na `nil` ter prilagoditve tipov `char` in `string` dolžine 1), le da se prevajanje nadaljuje.

Sintaksno drevo

Med najobsežnejšimi nalogami je bila implementacija razredov za gradnjo sintaksnega drevesa. Ker želimo, da si elementi drevesa delijo skupne atribute, smo s pridom izkoristili abstraktne razrede in dedovanje. Osnovni abstraktni razred, iz katerega so izpeljani vsi drugi razredi (elementi oziroma poddrevesa) sintaksnega drevesa, se imenuje `SynTree` in ima obliko:

```
public abstract class SynTree {

 /* Vrstica in stolpec, v katerem se začne del izvorne kode,
   ki ga opisuje to sintaksno drevo. */
 int line;
 int column;

 /* Konstruktor. */
 public SynTree(int line, int column) {
  this.line = line;
  this.column = column;
 }

 ...

 /* Metoda, ki opravi semantično analizo in pri tem izračuna
  vrednosti atributov. @return true, če pri izračunu vrednosti
  atributov ni napak; false sicer. */
 public abstract boolean semAnal();

 /* Prevajanje v vmesno kodo. */
 public abstract void translate();
}
```

Izvorna koda 4.4: SynTree.java

Vsak element sintaksnega drevesa ima najmanj dva atributa (vrstico in stolpec, ki označujeta njegov položaj v izvorni kodi), konstruktor ter metodi s katerima sprožimo semantično analizo tega dela izvorne kode in prevajanje v vmesno kodo.

Za primer poglejmo še dva razreda - `SynProgram` in `SynForStmt`, ki implementirata ta abstraktni razred:

```
public class SynProgram extends SynDecl
 /* SynDecl extends SynTree */ {

 /* Ime programa. */
 public SynIdentifier name;
 /* Deklaracije konstant. */
 public LinkedList<SynConstDecl> constDecls;
 /* Deklaracije tipov. */
 public LinkedList<SynTypeDecl> typeDecls;
 /* Deklaracije spremenljivk. */
 public LinkedList<SynVarDecl> varDecls;
 /* Deklaracije podprogramov. */
 public LinkedList<SynSubDecl> subDecls;
 /* Sestavljeni stavek programa. */
 public SynCompoundStmt stmt;

 /* Konstruktor. */
 public SynProgram(
   SynIdentifier name,
   LinkedList<SynConstDecl> constDecls,
   LinkedList<SynTypeDecl> typeDecls,
   LinkedList<SynVarDecl> varDecls,
   LinkedList<SynSubDecl> subDecls,
   SynCompoundStmt stmt) {
  super(name.line, name.column);
  this.name = name;
  this.constDecls = constDecls;
  this.typeDecls = typeDecls;
  this.varDecls = varDecls;
  this.subDecls = subDecls;
  this.stmt = stmt;
 }

 ...
}
```

Izvorna koda 4.5: SynProgram.java

Program je sestavljen iz imena, deklaracij konstant, tipov, spremenljivk in podprogramov ter sestavljenega stavka. `SynProgram` zato vsebuje dodatne atribute - reference na te elemente.

Stavek `for` opišemo z imenom spremenljivke, spodnjo in zgornjo mejo (ki sta lahko tudi izraza) ter telesom stavka. `SynForStmt` kot atribute vsebuje reference na te elemente:

```
public class SynForStmt extends SynStmt
 /* SynStmt extends SynTree */ {

 /* Ime spremenljivke. */
 public SynExprIdentifier var;
 /* Spodnja meja. */
 public SynExpr loBound;
 /* Zgornja meja. */
 public SynExpr hiBound;
 /* Telo. */
 public SynStmt stmt;

 /* Konstruktor. */
 public SynForStmt(
   int line, int column,
   SynExprIdentifier var,
   SynExpr loBound, SynExpr hiBound,
   SynStmt stmt) {
  super(line, column);
  this.var = var;
  this.loBound = loBound;
  this.hiBound = hiBound;
  this.stmt = stmt;
 }

 ...
}
```

Izvorna koda 4.6: SynForStmt.java

Podobno velja tudi za ostale razrede sintaksnega drevesa. Posebnost so še deklaracije, izrazi in podatkovni tipi, ki imajo dodaten atribut - semantični tip.

```
public SemType type;
```

Izvorna koda 4.7: SynTypeDecl.java

Semantična analiza

Semantično analizo programa (v našem primeru gre predvsem za preverjanje deklaracij, povezovanje imen z njihovim pomenom in preverjanje tipov) sprožimo z naslednjim ukazom nad sintaksnim drevesom:

```
boolean success = synTree.semAnal();
```

Izvorna koda 4.8: Main.java (semantična analiza)

Kot rezultat dobimo `true`, če se je semantična analiza izvedla brez napak, ter `false`, če je prišlo do kritičnih napak, zaradi katerih nadaljnje prevajanje ni mogoče. Na nekaj primerih si poglejmo, kako izgleda semantična analiza dela kode.

Pri enostavnih tipih zgolj shranimo semantični tip in vrnemo `true`:

```
public class SynStringType extends SynType {
 @Override
 public boolean semAnal() {
  type = new SemString();
  return true;
 }
}
```

Izvorna koda 4.9: SynStringType.java

Semantično analizo stavkov izvedemo tako, da rekurzivno preverimo semantično analizo vseh elementov ter vrnemo `true`, če pri tem nismo naleteli na napake. Na primeru `while` zanke vidimo, da preverimo izraz v pogoju in telo stavka. Dodatno zahtevamo, da se mora izraz v pogoju strukturno prilagoditi tipu `boolean`.

```
public class SynWhileStmt extends SynStmt {
 @Override
 public boolean semAnal() {
  if(!cond.semAnal())
   return false;

  if(!(cond.exprType.matchStructure(new SemBoolean())))
   Report.error("Condition not boolean.", line, column,1); //exit

  if(!body.semAnal())
   return false;

  return true;
 }
}
```

Izvorna koda 4.10: SynWhileStmt.java

Semantična analiza izrazov združuje zgornja pristopa, saj moramo semantično preveriti vse elemente ter shraniti semantični tip izraza. Na primeru izraza z

unarnim operatorjem, najprej izvedemo semantično analizo notranjega izraza, preverimo če operatorju `ADD` ali `SUB` sledi tip `integer` oziroma operatorju `NOT` tip `boolean`, na koncu pa še celotnemu izrazu določimo tip, ki je enak tipu notranjega izraza:

```
public class SynUnExpr extends SynExpr {
 @Override
 public boolean semAnal() {
  if(!expr.semAnal())
   return false;

  switch(oper){
   case ADD:
   case SUB:
    if(!(expr.exprType.matchStructure(new SemInteger())))
     Report.error("Unary operator 'ADD/SUB'
        not followed by integer!", line, column,1); //exit
    break;

   case NOT:
    if(!(expr.exprType.matchStructure(new SemBoolean())))
     Report.error("Unary operator 'NOT'
       not followed by boolean!", line, column,1); //exit
    break;
  }

  exprType=expr.exprType;
  return true;
 }}
```

Izvorna koda 4.11: SynUnExpr.java

Programi lahko uporabljajo standardno knjižnico, ki vsebuje nekatere uporabne procedure in funkcije, na primer za branje iz standardnega vhoda in pisanje na standardni izhod, alociranje in sproščanje pomnilnika ... Standardne knjižnice ne vključimo neposredno kot del programa, pač pa procedure in funkcije standardne knjižnice le deklariramo na globalnem nivoju, takoj ob začetku semantične analize.

Generiranje vmesne kode

Prevajanje semantično preverjenega sintaksnega drevesa sprožimo z ukazom:

```
synTree.translate();
```

Izvorna koda 4.12: Main.java (generiranje vmesne kode)

Najpreprostejša je pretvorba konstant:

```
public class SynNilConst extends SynConstExpr {
 /* Referenca na vmesno kodo. */
 public ImcExpr imc;

 @Override
 public void translate(){
  this.imc=new ImcCONST(0);
 }
}
```

Izvorna koda 4.13: SynNilConst.java

Še vedno preprosta je pretvorba izraza z binarnim operatorjem, kjer v vmesno kodo najprej pretvorimo levo in desno stran izraza, nato pa obe strani povežemo:

```
public class SynBinExpr extends SynExpr {
 @Override
 public void translate(){
  leftExpr.translate();
  rightExpr.translate();
  imc = new ImcBINOP(oper,leftExpr.imc,rightExpr.imc);
 }
}
```

Izvorna koda 4.14: SynBinExpr.java

Nekoliko zahtevnejša je pretvorba klicev funkcij, kjer je potrebno izračunati statično povezavo (gleda na nivo na katerem je funkcija definirana) ter prevesti in dodati argumente:

```
public class SynFuncCall extends SynExpr {
 @Override
 public void translate(){
  LinkedList<ImcExpr> list= new LinkedList<ImcExpr>();
  int diff=calling_frame.level-this_frame.level;

  //izracunamo in dodamo staticno povezavo
  ImcExpr expr=new ImcTEMP(calling_frame.FP);
  if(diff>=0){
   for(int i=0; i<=diff; i++){
    expr=new ImcMEM(expr);
   }
  }
  list.addLast(expr);
```

```
  //dodamo argumete
  for(SynExpr arg : args){
   arg.translate();
   list.addLast(arg.imc);
  }

  imc = new ImcFUNC(this_frame.entry, list);
 }
}
```

Izvorna koda 4.15: SynFuncCall.java

Vmesno kodo pretvorimo v kanonična drevesa, skozi proces semantične analize in pretvorbe v vmesno kodo pa sproti gradimo seznam "koščkov kode". Koščke kode ločimo na deklaracije znakovnih (`string`) konstant, deklaracije spremenljivk ter ostalo kodo. Seznam koščkov kode nam bo prišel prav v naslednjem koraku.

Generiranje ciljne kode

Ko smo uspešno implementirali dosedanje dele prevajalnika, nastopi zaključni korak: prevajanje v ciljno kodo.

Ciljno kodo razdelimo na dva dela - v prvem delu, ki ga napoveduje beseda `.text` se nahaja koda procedur, funkcij in sestavljenega stavka programa, drugi del, označenem z besedo `.data`, pa je namenjen rezervaciji pomnilnika za spremenljivke in znakovne konstante. Ker smo že v prejšnjih korakih sestavili sezname koščkov kode, je osnovna logika preprosta:

```
try {
 PrintStream codes = new PrintStream("code.mms");

 codes.println("\t.text");
 for (CodeChunk codeChunk : codeChunks)
  codeChunk.printMMIX(codes);
 codes.println("\n\t.data");
 for (VarChunk varChunk : varChunks)
  varChunk.printMMIX(codes);
 for (ConstChunk constChunk : constChunks)
  constChunk.printMMIX(codes);

 codes.close();
}
catch (Exception ex) {}
```

Izvorna koda 4.16: Main.java (generiranje ciljne kode)

Generiranje MMIX kode za spremenljivke in znakovne konstante je relativno preprosto. Za spremenljivke le rezerviramo potreben prostor v pomnilniku:

```
public class VarChunk extends Chunk {
 /* Naslov, na katerem je shranjena spremenljivka. */
 public Label label;

 /* Velikost spremenljivke. */
 public Integer size;

 @Override
 public void printMMIX(PrintStream file) {
  file.println(label.name+":\t.skip "+size);
 }
}
```

Izvorna koda 4.17: VarChunk.java

Znakovne konstante definiramo s tremi ukazi - poravnanje, navedbo dolžine ter zapisom same znakovne konstante:

```
public class ConstChunk extends Chunk {
 /* Naslov, na katerem je shranjena konstanta. */
 public Label label;

 /* Niz znakov. */
 public String value;

 @Override
 public void printMMIX(PrintStream file) {
  file.println("\t.align 8");
  file.println(label.name+":\tOCTA "+value.length());
  file.println("\tBYTE \""+value+"\"");
 }
}
```

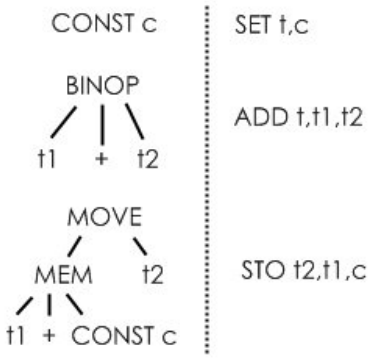
Izvorna koda 4.18: ConstChunk.java

Precej več dela je pri prevajanju preostale vmesne kode (funkcije, procedure in sestavljeni stavek programa). V prvem koraku kanonična drevesa vmesne kode, ki pripadajo temu delu kode (*chunk*), pretvorimo v ukaze v zbirniku za MMIX (nekaj vzorcev za pretvorbo prikazuje slika 4.1). Pri tem ukazi še vedno uporabljajo začasne spremenljivke, ki jih je potrebno zamenjati z ustreznimi registri (procesorji večino operacij še vedno opravljajo zgolj nad registri). Postopku zamenjave začasnih spremenljivk s procesorskimi registri pravimo tudi dodeljevanje registrov. Razširjen pristop k dodeljevanju registrov

je barvanje interferenčnega grafa, ki je uporabljen tudi v našem primeru. Ko so registri dodeljeni, je vsakemu delčku kode potrebno dodati prolog in epilog (primer: pri funkcijah v prologu pripravimo klicni zapis in shranimo povratni naslov, v epilogu pa povrnemo sklad ter nastavimo povratni naslov in rezultat), nato je del kode pripravljen na izpis.

```
public class CodeChunk extends Chunk {
 /* Seznam kanonicnih dreves vmesne kode. */
 public LinkedList<ImcStmt> stmts;
 /* Seznam strojnih ukazov. */
 public LinkedList<AsmInstr> instrs;

 /* Generiranje strojnih ukazov. */
 public void translate(){
  instrs = new LinkedList<AsmInstr>();
  for(ImcStmt stmt : stmts){
   stmt.translate(instrs);
  }
 }

 /* Generiranje strojnih ukazov, dodeljevanje registrov,
   dodajanje prologa in epiloga ter izpis MMIX kode. */
 @Override
 public void printMMIX(PrintStream file) {
  translate();

  Interference ig=new Interference(this);
  ig.regAlloc(AsmInstr.stReg);

  addPrologEpilog(ig.getMap());

  file.println("\t.global "+frame.entry.name);
  for(AsmInstr instr:instrs){
   String koda=instr.formatMMIX(ig.getMap());
   if(koda!=null)
    file.println(koda);
  }
 }
}
```

Izvorna koda 4.19: CodeChunk.java



Slika 4.1: Vzorci za pretvorbo dreves vmesne kode v MMIX zbirnik

Primer izpisane kode v zbirniku:

```
    .text
    .global _main
_main:
    # prologue ...
    STO $3,$4,0
    SETL $0,0
    STO $0,$4,8
    SETL $0,5
    STO $0,$4,16
    PUSHJ $0,L1
    ADD $0,$2,0
    GET A $1,V1
    STO $0,$1,0
    # epilogue ...

    .data
V1: .skip 8
V2: .skip 32
    .align 8
S1: OCTA 3
    BYTE "abc"
```

Izvorna koda 4.20: code.mms

4.3 Prilagoditev za izvajanje v oblaku

Prevajalnik, opisan v razdelku 4.2, bo služil kot osnova za različico, ki bo tekla v oblaku.

Za začetek si iz uradne spletne strani platforme Google App Engine [14] na lokalni računalnik prenesemo in namestimo razvojni paket App Engine SDK za programski jezik Java, ki vključuje knjižnice za GAE API vmesnike, razvojni strežnik ter orodja za prenos (*upload*) izdelane aplikacije na platformo GAE. Če pri programiranju uporabljamo razvojno okolje Eclipse, pa samo namestimo Google vtičnik za Eclipse, katerega del je tudi razvojni paket. V tem primeru se nekatera opravila znatno olajšajo, na primer priprava novega projekta, namenjenega za platformo GAE (samodejno se ustvari zahtevana struktura map ter opisne xml datoteke), uporaba testnega strežnika (izberemo *debug* aplikacije na testnem strežniku) ter prenos aplikacije na platformo GAE.

JSP

Ker bo aplikacija dostopna preko spletnega brskalnika, je smiselno pripraviti nekaj spletnih strani, ki bodo služile kot vstopna točka in predstavitev aplikacije. Zelo uporabne so JSP strani, ki so tudi podprte s strani GAE ter lahko z njimi prikazujemo dinamične spletne strani. Enostavna JSP stran, ki bo služila kot vstopna točka za spletni prevajalnik:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<jsp:include page="header.jsp"/>
 <h3>Prevajalnik</h3>
 <p>Pošlji izvorno kodo:</p>
 <p><form method="post" action="/compiler"
          enctype="multipart/form-data">
  <div><input type="file" name="source">
       <input type="submit" value="Pošlji"/></div>
  </form></p>
 <h4>ALI</h4>
 <p>izberi primer:</p>
 <p><ul>
  <li><a href="/compiler?file=demo1">demo1</a></li>
  <li><a href="/compiler?file=demo2">demo2</a></li>
  <li><a href="/compiler?file=demo3">demo3</a></li>
 </ul></p>
<jsp:include page="footer.jsp"/>
```

Izvorna koda 4.21: compiler.jsp

V zgornjo spletno stran je vključena glava *header.jsp*, ki se poslužuje GAE API vmesnika za dostop do podatkov o trenutnem uporabniku. Če je uporabnik prijavljen v Google račun, mu lahko za začetek izpišemo osebno pozdravno sporočilo, resnično uporabna pa je možnost prilagajanja aplikacije posameznemu uporabniku, ki pa tukaj ni uporabljena, saj prevajalnik enako obravnava vse uporabnike.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="com.google.appengine.api.users.*" %>

<%
UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
if (user != null) {
%>
 <p class="header">Pozdravljen/a, <%= user.getNickname() %>!
 (<a href="<%= userService.createLogoutURL(
                request.getRequestURI()) %>">odjava</a>)</p>
<%
}else {
%>
 <p class="header">Pozdravljen/a!
 (<a href="<%= userService.createLoginURL(
                request.getRequestURI()) %>">prijava</a>)</p>
<%
}
%>
```

Izvorna koda 4.22: header.jsp

Nastavitvene datoteke GAE

Ko uporabnik naloži spletno stran *compiler.jsp* ima na voljo dve možnosti: lahko pošlje svojo datoteko z izvorno kodo v prevajanje, ali iz seznama izbere primer. Iz izvorne kode 4.21 je razvidno, da je zahtevek za prevajanje poslan na naslov */compiler*. Da bi GAE vedel, kam zahtevek usmeriti, moramo ustrezno popraviti nastavitvene datoteke v mapi *war/WEB-INF*. Bistveni del se skriva v datoteki *web.xml*. V našem primeru je vsebina datoteke naslednja:

```
<?xml version="1.0" encoding="utf-8"?><web-app>
<servlet>
 <servlet-name>compiler</servlet-name>
 <servlet-class>diploma.servlet.CompilerServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
 <servlet-name>compiler</servlet-name>
 <url-pattern>/compiler</url-pattern>
</servlet-mapping>
<servlet>
 <servlet-name>savefile</servlet-name>
 <servlet-class>diploma.servlet.SaveFileServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>savefile</servlet-name>
 <url-pattern>/savefile</url-pattern>
</servlet-mapping>
<welcome-file-list>
 <welcome-file>compiler.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

Izvorna koda 4.23: web.xml

Datoteko *logging.properties* pustimo s privzetimi nastavitvami, *appengine-web.xml* pa dopolnimo s podatki o aplikaciji (ID, verzija) in statičnih datotekah:

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
 <application>hellomycloud</application>
 <version>1</version>
 <system-properties>
  <property name = "java.util.logging.config.file"
            value = "WEB-INF/logging.properties"/>
 </system-properties>
 <static-files>
  <include path="/examples/**.pas" />
 </static-files>
</appengine-web-app>
```

Izvorna koda 4.24: appengine-web.xml

Java Servlet

Zahtevek za prevajanje je torej usmerjen na servlet *CompilerServlet*, ki bo imel podobno vlogo, kot jo ima *Main.java* v osnovnem prevajalniku. Kako ločiti, ali je bila poslana datoteka z izvorno kodo ali je bil izbran primer? Enostavno, v prvem primeru imamo zahtevo HTTP POST, v drugem pa HTTP GET, ki ju servlet enostavno loči. Potrebno je le pravilno prebrati vhodno izvorno datoteko, nadaljnje prevajanje je enako.

```
public class CompilerServlet extends HttpServlet {

 // Ta metoda se izvede, ko uporabnik izbere enega izmed primerov.
 public void doGet(HttpServletRequest req, HttpServletResponse res)
     throws ServletException, IOException {

  /* Pripravimo izpis. */
  res.setContentType("text/html");
  HtmlWriter htmlWriter = new HtmlWriter(res.getWriter());

  /* Odpremo vhodno datoteko z izvorno kodo. */
  String filename, serverURL = "http://.../examples/";
  BufferedReader src;
  try {
   filename = req.getParameter("file");
   URL srcURL = new URL(serverURL+filename+".pas");
   src = new BufferedReader(
             new InputStreamReader(srcURL.openStream()));
   log.info("Example file '" +filename+ "' opened successfully.");
  } catch (FileNotFoundException ex) {
   log.warning(ex.toString());
   src = null;
  }

  // Izvedemo prevajanje ...
  compile(src, htmlWriter, filename);
 }


 // Ta metoda se izvede, ko uporabnik pošlje svojo datoteko.
 public void doPost(HttpServletRequest req,HttpServletResponse res)
      throws ServletException, IOException {

  /* Pripravimo izpis. */
  res.setContentType("text/html");
  HtmlWriter htmlWriter = new HtmlWriter(res.getWriter());

  /* Preberemo vhodno datoteko, ki vsebuje izvorno kodo. */
  String filename;
  BufferedReader src = null;
  try {
   ServletFileUpload upload = new ServletFileUpload();
   FileItemIterator iterator = upload.getItemIterator(req);
   while (src == null && iterator.hasNext()) {
    FileItemStream item = iterator.next();
    InputStream stream = item.openStream();
```

```
    if (!(item.isFormField()) &&
       item.getFieldName().equalsIgnoreCase("source")) {
     src = new BufferedReader(new InputStreamReader(stream));
     filename = item.getName();
     log.info("Got a file: " + filename);
    }
   }
  }catch (Exception ex) {
   log.warning(ex.toString());
   src = null;
  }

  // Izvedemo prevajanje ...
  compile(src, htmlWriter, filename);
 }

}
```

Izvorna koda 4.25: CompilerServlet.java (prvi del)

Izpis sporočil

Nadaljnji postopek prevajanja je praktično enak kot pri navadnem prevajalniku, ena večjih razlik med namizno ter spletno različico prevajalnika, pa je v odsotnosti ukazne vrstice oziroma standardnega vhoda/izhoda. Vsa sporočila *System.out* strežniki GAE namreč ujamejo ter so uporabniku nevidna. Poiskati je treba drugačno pot in vsa sporočila izpisati v obliki HTML. V ta namen sem razvil razred *HtmlWriter*, ki je kot parameter podan v proces prevajanja. *HtmlWriter* je v bistvu zelo preprost razred, saj je vse kar počne zgolj to, da izpisuje sporočila v HTML obliki na izhodni tok HTML odgovora (*response*), pri čemer napake obarva rdeče, opozorila oranžno ter obvestila o uspehu zeleno.

```
public class HtmlWriter {

 private PrintWriter out;

 public HtmlWriter (PrintWriter out){
  this.out = out;
 }

 /** Izpiše sporočilo.
  * @param msg Sporočilo. */
 public void printbr(String msg) {
```

```
  out.print(msg + "<br/>");
 }

 /** Izpiše informacijo o uspehu
  * @param msg Sporocilo. */
 public void success(String msg) {
  out.print("<span style=\"color:green;\">");
  out.println(":-) " + msg);
  out.print("</span><br/>");
 }

 /** Izpiše opozorilo, vezano na vrstico in stolpec izvorne kode.
  * @param msg Opozorilo.
  * @param line Vrstica izvorne kode.
  * @param column Stoplec izvorne kode. */
 public void warning(String msg, int line, int column) {
  out.print("<span style=\"color:orange;\">");
  out.print(":-o [" + line + ":" + column + "] " + msg);
  out.print("</span><br/>");
 }

 /** Izpiše obvestilo o napaki in prekine izvajanje.
  * @param msg Obvestilo o napaki. */
 public void fatalError(String msg) {
  out.print("<span style=\"color:red;\">");
  out.print(":-( " + msg);
  out.print("</span><br/>");
  throw new InternalError("Error(s) exist(s) in source code.");
 }

 /** Izpiše obvestilo o napaki in prekine izvajanje.
  * @param msg Obvestilo o napaki.
  * @param line Vrstica izvorne kode.
  * @param column Stolpec izvorne kode. */
 public void fatalError(String msg, int line, int column) {
  out.print("<span style=\"color:red;\">");
  out.print(":-( [" + line + ":" + column + "] " + msg);
  out.print("</span><br/>");
  throw new InternalError("Error(s) exist(s) in source code.");
 }
}
```

Izvorna koda 4.26: HtmlWriter.java

HtmlWriter mora biti posredovan vsem metodam in razredom, ki želijo izpisati kakršno koli sporočilo uporabniku:

```
/* Glavna metoda prevajanja iz FRI-Pascal v MMIX. */
private void compile(BufferedReader src, HtmlWriter htmlWriter,
          String filename) {
 try{
  /* Leksikalna in sintaktična analiza. */
  SynTree synTree = lexSynAnalysis(src, htmlWriter);
  /* Semantična analiza. */
  if (synTree!=null){
   if (synTree.semAnal(htmlWriter)) {
    htmlWriter.success("Semantic analysis found no errors.");
    /* Prevajanje v vmesno kodo. */
    synTree.translate(htmlWriter);
    /* Izpis MMIX kode. */
    writeMMIX(htmlWriter, filename);
   } else {
    htmlWriter.error("Error(s) exist(s) in source code.");
   }
  }
 }catch (Exception e) {
  htmlWriter.error("Compiling failed.");
  log.warning(e.getMessage());
 }catch (InternalError e) {
  htmlWriter.error("Compiling failed.");
  log.warning(e.getMessage());
 }
}

/* Izračun sintaksnega drevesa (leksikalna, sintaktična analiza).*/
private SynTree lexSynAnalysis(BufferedReader src,
                               HtmlWriter htmlWriter) {
 SynTree synTree = null;
 PascalLex lexer = new PascalLex(src, htmlWriter);
 PascalSyn parser = new PascalSyn(lexer, htmlWriter);
 try {
  synTree = (SynProgram)(parser.parse().value);
  htmlWriter.success("Lexical/syntax analysis found no errors.");
 } catch (Exception ex) {
  htmlWriter.error("Error(s) exist(s) in source code.");
  log.warning(ex.getMessage());
 }
 return synTree;
}
```

Izvorna koda 4.27: CompilerServlet.java (drugi del)

Izpis MMIX kode

Zaključni korak uspešnega prevajanja je izpis MMIX kode. Zaradi lepšega izpisa in kasnejšega shranjevanja, kodo ovijemo v tekstovno okno obrazca (*form*). Uporabnik lahko ciljno kodo pregleda, nato pa datoteko z izvorno kodo po želji s klikom na gumb prenese:

```
/** Izpis v zbirnik za MMIX. */
private void writeMMIX(HtmlWriter htmlWriter, String filename) {
 try {
  htmlWriter.printbr("<div class=\"white_box\">MMIX code:");
  htmlWriter.print("<form method=\"post\" action=\"/savefile\" >");
  htmlWriter.print("<textarea name=\"code\" rows=\"40\"
                              cols=\"57\" readonly >");

  htmlWriter.println("\t.text");
  for (CodeChunk codeChunk : Chunk.codeChunks) {
   htmlWriter.println("");
   codeChunk.printMMIX(htmlWriter);
  }

  htmlWriter.println("\n\t.data");
  for (VarChunk varChunk : Chunk.varChunks)
   varChunk.printMMIX(htmlWriter);
  for (ConstChunk constChunk : Chunk.constChunks)
   constChunk.printMMIX(htmlWriter);

  htmlWriter.print("</textarea>");
  htmlWriter.print("<input type=\"hidden\" name=\"filename\"
                           value=\"" + filename + ".mms\">");
  htmlWriter.print("<div><input type=\"submit\"
                         value=\"Save file...\" /> </div> </form>");
  htmlWriter.print("</div> <br/> <div class=\"white_box\">");
 }
 catch (Exception ex) {
  log.warning(ex.toString());
 }
}
```

Izvorna koda 4.28: CompilerServlet.java (tretji del)

```
public class SaveFileServlet extends HttpServlet {
 /*Klic te metode sproži prenos kode, ki je podana kot parameter.*/
 public void doPost(HttpServletRequest req,HttpServletResponse res)
                throws ServletException, IOException {
  String mmixcode = req.getParameter("code");
  String filename = req.getParameter("filename");
```

```
  res.setHeader("Content-Disposition",
                "attachment; filename=\""+filename+"\"");
  res.setContentType("text/plain");
  PrintWriter out = res.getWriter();
  out.write(mmixcode);
  out.close();
 }
}
```

Izvorna koda 4.29: SaveFileServlet.java

Dodatki

Dodatna zmožnost prevajalnika je izpis sintaksnega drevesa, vmesne kode ter koščkov kode v XML obliki. Ti izpisi so bolj kot za nadaljnjo uporabo koristni pri preverjanju pravilnega delovanja prevajalnika

Zato pa je toliko bolj zanimivo **interpretiranje** koščkov kode. Interpreter simulira izvajanje programa in vrne vrednost prve definirane spremenljivke v programu. Ali naš program vrača razumljive rezultate, lahko torej enostavno preverimo tako, da takoj na začetku programa deklariramo spremenljivko, v kateri bomo vrnili rezultat, nato pa nam interpreter že med procesom prevajanja izpiše rezultat. Omeniti pa je potrebno pomembno omejitev: interpreter ne podpira uporabe standardne knjižnice. To je razumljivo, saj interpreter nima dostopa do standardnega vhoda in izhoda, prav tako ne more upravljati s pomnilnikom. Celotno izvorno kodo interpreterja je razvil doc. dr. Boštjan Slivnik.

4.4 Težave in rešitve

Uporaba platforme GAE je solidno dokumentirana, zato večjih težav nisem imel. Nekaj preglavic mi je povzročalo le dejstvo, da je storitev še v razvoju ter se aktivno spreminja. V času izdelave aplikacije je med drugim podpora programskemu jeziku Java izgubila oznako zgodnji predogled (*early preview*).

Osnovni prevajalnik je napake izpisoval v konzolo, nato pa izvajanje prekinil z ukazom `System.exit(1)`. Platforma GAE vključuje nekaj omejitev in med njimi je tudi prepoved uporabe oziroma onemogočenje nekaterih metod iz razreda `System`, vključno z `exit()`. Klic omenjene metode v spletni različici prevajalnika je povzročil sesutje izvajanja in prikaz obvestila o `java.security.AccessControlException`. Logična rešitev je vključila lovljenje (*catch*) omenjene izjeme. Izvajanje se ni več grobo prekinilo, ampak je

servlet izpisal sporočilo o napaki in zaključil izvajanje. S tem pa težav ni bilo konec. Z novo verzijo platforme in razvojnega paketa, klic `System.exit(1)` ni več sprožil omenjene izjeme, ampak nas je v brskalniku pričakalo obvestilo o napaki strežnika. Potrebno je bilo poiskati drugačno rešitev. Vse klice `System.exit(1)` sem zamenjal s `throw new InternalError()` ter ustrezno prilagodil *catch* stavek.

4.5 Možnosti izboljšav

Prevajalnik je implementiran, postavljen v oblak in svojo osnovno nalogo (prevajanje iz jezika miniPascal v zbirni jezik za arhitekturo MMIX) opravlja. Še vedno pa je veliko prostora za izboljšave.

Pozoren bralec je opazil, da smo pri izvedbi izpustili pomemben korak analize in optimizacije vmesne kode. Optimizacija vmesne kode bi doprinesla k zmanjšanju števila MMIX ukazov ter posledično k hitrejšemu delovanju. Prav tako obstajajo različni pristopi, kako sintaksno drevo prevajamo v vmesno kodo ter vmesno kodo v ciljno kodo. Nekateri so pri generiranju optimizirane kode uspešnejši od drugih.

Celotno delovanje prevajalnika bi lahko pohitrili z zmanjšanjem števila klicev metod in konstruktorjev, z doslednim sproščanjem pomnilnika, odpravljanjem nepotrebnih pogojnih stavkov in zank ter drugimi pristopi.

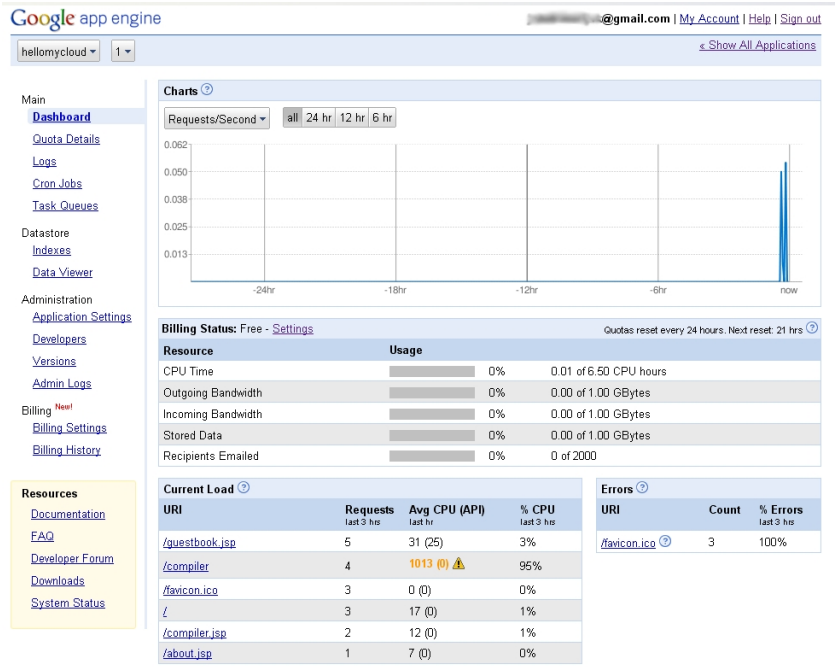
Prevajalnik je uporaben le v primeru, ko se drži specifikacij izvornega in ciljnega jezika. Ob morebitnih spremembah v specifikacijah, bi bilo potrebno prevajalnik ustrezno prilagoditi. Nenazadnje: leksikalna, sintaksna in semantična pravila bi lahko prilagodili originalnemu programskemu jeziku Pascal. Le-ta je (bil) nedvomno bolj razširjen kot miniPascal.

4.6 Nadzorna plošča, verzije, opravila

Ko smo našo aplikacijo uspešno postavili na uporabo širnemu svetu, se lahko prijavimo v nadzorno ploščo (omenjeno že v razdelku 3.2) in spremljamo, kaj se z aplikacijo dogaja. Na ogled so nam dnevniške datoteke (log), pregled porabe prostora in procesorskega časa, upravljanje z verzijami aplikacije, pregled objektov v bazi in drugo.

GAE podpira poganjanje različnih verzij aplikacije. Tako lahko sočasno uporabnikom strežemo stabilno verzijo, obenem pa preizkušamo novo, razvojno različico. Verzija, označena kot privzeta, je na voljo na naslovu *applicationid*.appspot.com, preostale pa so dosegljive na *id*.latest.*applicationid*.appspot.com.

Na primeru razvitega prevajalnika, je na spletnem naslovu http://hellomycloud.appspot.com dosegljiva osnovna različica, na naslovu http://2.latest.hellomycloud.appspot.com pa različica, ki pri prevajanju izpisuje tudi sintaksno drevesa, vmesno kodo in koščke kode v XML obliki ter vključuje interpreter.



Slika 4.2: Nadzorna plošča Google App Engine.

Dodatna možnost je določanje opravil, ki se samodejno izvedejo na določen termin. Primer datoteke, ki definira opravilo, ki se izvede vsako nedeljsko noč:

```
<?xml version="1.0" encoding="UTF-8"?>
<cronentries>
  <cron>
    <url>/schedule</url>
    <description>sunday night test</description>
    <schedule>every sunday 04:00</schedule>
    <timezone>Europe/Ljubljana</timezone>
  </cron>
</cronentries>
```

Izvorna koda 4.30: cron.xml

V klicanem servletu bi, na primer, lahko s SVN strežnika prenesli zadnjo različico izvorne kode ter avtomatizirali redno prevajanje v kodo v zbirniku za MMIX.

Poglavje 5

Sklep

Dosedanje izkušnje so mešane. Slišati je velike zgodbe o uspehu, številna pa so tudi razočaranja. Kritike letijo predvsem na nedokončanost storitev, občasne izpade, slabšo odzivnost ob večjih obremenitvah ter pomanjkanje standardov.

Kot vsako novo (oziroma na novo odkrito) področje je tudi računalništvo v oblaku včasih megleno, neoprijemljivo, nedozorelo. Prve storitve so (bile) "surove", zahtevajo več poznavanja in znanja za upravljanje. S časom in pojavom konkurence pa se vedno bolj izpopolnjujejo in postajajo bolj in bolj prijazne do uporabnika. Nekdo se je slikovito izrazil: "V letu 2007 je bilo področje računalništva v oblaku kot novi svet, leta 2008 kot kot divji zahod, sedaj pa se počasi kolonizira."[6] Obenem računalništvo v oblaku vpliva tudi na ostale veje industrije, od proizvajalcev čipov, snovanja in trženja strojne in programske opreme, do zmanjšanja obsega naročil razvoja programske opreme pri zunanjih izvajalcih (*outsourcing*).

Računalništvo v oblaku predstavlja **izziv** in **priložnost**.

5.1 Izzivi

V bližnji prihodnosti čakajo računalništvo v oblaku nekateri pomembni izzivi.

Razveseljivo je, da že sedaj veliko vlogo pri razvoju računalništva v oblaku igrajo odprti standardi. Veliko storitev temelji na tehnologijah in standardih kot so XMPP, AJAX, XML, JSON, REST. Za dokončen uspeh, pa bo treba poskrbeti tudi za **standardizacijo** - zagotavljanje združljivosti in prenosljivosti. Z reševanjem te problematike se ukvarja projekt Deltacloud, katerega glavnik podpornik je Red Hat.

Pomembno je zagotoviti visok **nivo upravljanja s podatki**. Le tako bodo uporabniki prepričani, da je hramba podatkov vredna zaupanja, zaščitena pred ne-

pooblaščenim dostopom, varna v primeru katastrof. V izogib izgubi podatkov ali primerom, ko do njih ne moramo dostopati, se priporoča vnaprejšnje preverjanje ponudnikove politike do te problematike. Pogovorili naj bi se predvsem o tem kdo ima privilegiran dostop do podatkov, o skladnosti z zakonodajo, lokaciji podatkov (ali lahko določimo, kje se hranijo), možnosti šifriranja podatkov, obnovi podatkov po nesrečah (ali zagotavljajo popolno obnovitev podatkov in koliko časa traja), preiskavi nepravilnih in nelegalnih aktivnosti, kaj bi se zgodilo s podatki, če podjetje propade, možnost popolne premestitve podatkov. V praksi je najbolje ponudnika preveriti s praktičnim preizkusom.

Izboljšati je potrebno splošno **varnost** ter varnostno upravljanje virtualiziranih sistemov. Zmotno je mnenje, da sta virtualizacija in nadzornik navideznih strojev zadostno zagotovilo za varnost. Če je nekaj virtualizirano, še ni samo po sebi varno.

Naslednji izziv bo **ločevanje zrnja od plevela**. Ob naraščajoči priljubljenosti se pojavlja ogromno storitev, tudi slabih, zastavljenih z namenom izkoriščanja popularnosti in pridobivanje lahkega zaslužka. Te storitve bodo metale slabo luč na celotno področje. Boljše storitve bodo zagotovile zanesljivost in zadosten nivo storitve (QoS) in se ločile tudi po dodanih storitvah, kot so samodejno skaliranje (*avtoscaling*), podpora poslovni inteligenci in poslovnim pravilom ter upravljanje poslovnega procesa.

Za polno **izkoriščanje prednosti**, ki jih ponuja računalništvo v oblaku, ni dovolj, da zgolj prenesemo naše aplikacije iz lokalnega strežnika v splet. Priporočljivo je, da aplikacije že gradimo z mislijo na izvajanje v oblaku. Le tako bomo izkoristili prednosti arhitektur računalništva v oblaku. [7]

Kako na globalnem trgu zagotoviti **skladnost z zakonodajo**, ki se od države do države razlikuje? Nekateri ponudniki poskušajo z ločenimi podatkovni centri (trenutno v ZDA in Evropski uniji) ter omejitvijo uporabe glede na geografsko lokacijo.

Spremenjenemu načinu uporabe programske opreme se bodo morale prilagoditi tudi **licenčne pogodbe**. Organizacija Free Software Foundation je že predstavila licenco *Affero General Public Licence*, namenjeno licenciranju odprtokodnih projektov, ki bodo uporabljeni za zagotavljanje storitev računalništva v oblaku.

Na koncu sledi še vprašanje: ali računalništvo v oblaku omejuje in zavira kreativnost ali spodbuja nove, inovativne pristope?

5.2 Prihodnost

Kar nekaj dejavnikov govori v prid napovedi, da se bo področje še naprej hitro razvijalo in pridobivalo na priljubljenosti. Naraščajoče potrebe po prostoru za hrambo podatkov in vedno zmogljivejšimi sistemi (še posebej na področju videa, simulacij, znanstvenih izračunov) prinaša stalno potrebo po dopolnjevanju in razširjanju obstoječih (lastnih) računalniških sistemov. Računalništvo v oblaku prinaša mamljivo alternativo, ki obljublja znatno zmanjšanje dela na tem področju, boljše razmerje

zmogljivost/cena in s tem nižje stroške, hitro (in navidez neomejeno) razširljivost, robustnost in lažji nadzor z zmogljivimi orodji za upravljanje. Ob vseprisotnih hitrih internetnih povezavah, običajno tudi hitrost prenosa podatkov ne predstavlja več omejujočega dejavnika. Ne smemo spregledati niti splošne miselnosti, ki čedalje bolj podpira pobude v smeri ekologije. Boljši izkoristek strojne opreme pomeni prihranke energije. Zgovorno je dejstvo, da se celo največja podjetja pripravljajo na "skok v oblak".

Računalništvo v oblaku pogostokrat primerjajo s pojavom električnih omrežij. Podjetja in tovarne so sprva imele vsaka svoj električni generator. Ob uvajanju javnega električnega omrežja so se gotovo pojavljali pomisleki glede zaupanja v zanesljivost in cenovno ugodnost ter strah pred odvisnostjo od ponudnika električne energije, pa tudi druga vprašanja. Sčasoma pa je postalo samoumevno, da je vsakdo priključen na električno omrežje, plačuje po dejanski porabi električne energije ter se znebi vseh drugih stroškov, povezanih s pridobivanjem električne energije. V podjetjih so namreč vedno težje opravičevali višjo ceno in vzroke izpadov lastne proizvodnje električne energije. Nasprotno se je trg ponudnikov električne energije širil, medsebojno povezovanje in standardi pa so poskrbeli za izredno visoko zanesljivost in ugodne cene. Zaradi podobnih razlogov, naj bi tudi računalništvo počasi prešlo k storitvenemu modelu.

Leta izkušenj so pokazala, da je v nekaterih primerih vseeno dobro imeti lastno rezervo. Pomislimo samo na bolnišnice, kjer si ne morejo privoščiti niti krajših izpadov električne energije, ker je marsikatero življenje dejansko odvisno od neprekinjene dobave električne energije! V takih primerih se probleme rešuje s pomočjo akumulatorjev in lastnih generatorjev električne energije. Če lahko naredimo analogijo: **koncept računalništva v oblaku bo sčasoma zadovoljil veliko večino potreb, kritične dejavnosti pa bodo verjetno še naprej poskrbele za izhod v sili - lastne rešitve**.

Dodatek A

Specifikacija jezika miniPascal

A.1 Leksikalna pravila

Programski jezik miniPascal uporablja ASCII abecedo in dosledno ločuje med veliki in malimi črkami. Znak za presledek (v kolikor ni del znakovne konstante), znak za skok v novo vrstico in tabulator ne pripadajo nobeni besedi.

1. **Imena:**
   (a) Ime programa, konstante, tipa, spremenljivke, procedure ali funkcije je poljuben niz, ki je sestavljen iz črk, števk in podčrtaja, in se začne s črko ali podčrtajem.
   (b) Niz, ki predstavlja ključno besedo, osnovni podatkovni tip ali logično konstanto, ni ime.
2. **Konstante:**
   (a) številčna konstanta je poljubno zaporedje števk.
   (b) Logični konstanti sta niza `true` in `false`.
   (c) Znakovna konstanta je (lahko tudi prazen) niz znakov ASCII abecede s kodami med (vključno) 32 in 126, zaprt v enojne navednice. Izjema je znak enojna navednica, ki mora biti znotraj znakovne konstane podvojen.
3. **Osnovni podatkovni tipi:**
   (a) Nizi `boolean`, `integer`, `char` in `string` so imena osnovnih podatkovnih tipov.

4. **Ključne besede:**

   (a) Ključne besede so:

   ```
   and array begin const div do else end for function
   if mod nil not of or procedure program record then
   to type var while
   ```

5. **Ostali simboli:**

   (a) Ostali osnovni simboli so:

   ```
   + - * = <> < > <= >= ( ) [ ] ^ . , : ; :=
   ```

6. **Komentarji:**

   (a) Začetek in konec komentarja označujeta znaka `{` in `}`, zaporedoma.

   (b) Komentarji so lahko vgnezdeni.

A.2 Sintaksna pravila

Sintaksna pravila sem namenoma zapisal v razmeroma neformalni obliki — upam, da dovolj formalno, da so (bolj ali manj) nedvoumna, hkrati pa dovolj neformalno, da ostaja zapis gramatike naloga.

1. **Program:**

   Program je sestavljen iz glave programa, deklaracij in sestavljenega stavka, ki mu sledi pika. Glavo programa uvaja ključna beseda `program` in je oblike

   `program` *identifier* `;`

2. **Deklaracije:**

   Deklaracije vsebujejo (v natančno tem vrstnem redu): deklaracije konstant, deklaracije tipov, deklaracije spremenljivk, deklaracije podprogramov. Deklaracija vsake od naštetih kategorij lahko manjka.

3. **Deklaracije konstant:**

   Deklaracije konstant uvaja ključna beseda `const` in so oblike

   `const` *identifier* `=` *const* `;`
   identifier `=` *const* `;`
   ⋮
   identifier `=` *const* `;`

Pri tem *const* predstavlja ime, (predznačeno) celo število, logično ali znakovno konstanto.

4. **Deklaracije tipov:**

   Deklaracije tipov uvaja ključna beseda `type` in so oblike

   `type` *identifier* `=` *type* `;`
   identifier `=` *type* `;`
   ⋮
   identifier `=` *type* `;`

   Pri tem *type* opisuje tip ali ime tipa.

   Tip je bodisi enostaven (`boolean`, `integer`, `char` ali `string`) ali sestavljen. Sestavljeni tipi so naslednji:

   (a) kazalec na podatkovni tip:

   `^` *type*

   (b) zapis:

   `record`
   identifier `:` *type* `;`
   identifier `:` *type* `;`
   ⋮
   identifier `:` *type*
   `end`

   Pri tem vsak zapis vsebuje vsaj eno polje.

   (c) tabela:

   `array [` *int-const* `] of` *type*

   Pri tem *int-const* označuje nepredznačeno celoštevilčno konstanto.

5. **Deklaracije spremenljivk:**

   Deklaracije spremenljivk uvaja ključna beseda `var` in so oblike

   `var` *identifiers* `:` *type* `;`
   identifiers `:` *type* `;`
   ⋮
   identifiers `:` *type* `;`

   Pri tem *identifiers* predstavlja seznam z vejicami ločenih imen, *type* pa opis tipa.

6. **Deklaracije podprogramov:**

   Deklaracije podprogramov vsebujejo poljubno zaporedje podprogramov. Vsak podprogram je sestavljen iz glave podprograma, deklaracij in sestavljenega stavka, ki mu sledi podpičje. Glavo podprograma uvaja bodisi ključna beseda `procedure` bodisi ključna beseda `function`, glava pa ima obliki

   `procedure` *identifier* `(` *parameters* `) ;`
   `function` *identifier* `(` *parameters* `) :` *type* `;`

   Pri tem *parameters* označuje (morebiti prazen) seznam parametrov:

   identifier `:` *type* `,` *identifier* `:` *type* `,` ... `,` *identifier* `:` *type*

7. **Stavki:**

   Stavek (*statement*) ima eno od naslednjih oblik:

   (a) prireditveni stavek:

   var-expression `:=` *expression*

   Pri tem *var-expression* opisuje referenco na pomnilnik.

   (b) sestavljeni stavek:

   `begin`
   statement `;`
   statement `;`
   ⋮
   statement
   `end`

   Pri tem je zaporedje stavkov lahko prazno.

   (c) pogojni stavek (dve obliki):

   `if` *expression* `then` *statement*
   `if` *expression* `then` *statement* `else` *statement*

   (`else`-del pogojnega stavka se vedno veže na najbližji `if`-stavek);

   (d) zanki:

   `while` *expression* `do` *statement*
   `for` *identifier* `:=` *expression* `to` *expression* `do` *statement*

   (e) klic procedure:

   identifier `(` *arguments* `)`

   Pri tem *argument* označuje (morebiti prazno) z vejicami ločeno zaporedje izrazov.

8. **Izrazi:**

   Izraz (*expression*) ima eno od naslednjih oblik:

   (a) enostaven izraz (`nil`, logična, celoštevilčna ali znakovna konstanta);

   (b) izraz v oklepajih:

   `(` *expression* `)`

   (c) izraz z unarnim operatorjem:

   unary-op expression

   Pri tem *unary-op* predstavlja `not`, `+` ali `-` .

   (d) izraz z binarnim operatorjem:

   expression binary-op expression

   Pri tem *binary-op* predstavlja:

   - `or`,
   - `and`,
   - `=`, `<>`, `<`, `>`, `<=`, `>=`,
   - `+`, `-`,
   - `*`, `div` in `mod`

   (`or` veže najšibkeje, `*`, `div` in `mod` najmočneje; primerjalni operatorji niso asociativni, ostali operatorji so levo asociativni);

   (e) klic funkcije:

   identifier `(` *arguments* `)`

   Pri tem *argument* označuje (morebiti prazno) z vejicami ločeno zaporedje izrazov.

   (f) referenco na pomnilnik (*var-expression*), ki ima eno od naslednjih oblik:

   identifier
   var-expression `^`
   var-expression `.` *identifier*
   var-expression `[` *expression* `]`

A.3 Semantična pravila

Podatkovni tipi

Vsaka deklaracija tipa (četudi gre zgolj za preimenovanje) ustvari nov tip. Natančneje, nova deklaracija tipa je bodisi vsak opis tipa, ki vsebuje ^ (kazalec), `array` ali `record`, bodisi ime tipa (kot "identifer"), če je uporabljen na desni strani enačaja v `type` deklaraciji.

Dva konstrukta se lahko tipsko prilagodita,

- če sta tipa obeh konstruktov imensko enaka, ali
- če sta tipa obeh konstruktov strukturno enaka in eden od obeh konstruktov opisuje izraz konstantne vrednosti.

Poleg tega se tip `char` (kot izvor) prilagodi tipu `string` (kot ponor) — vsiljena pretvorba podatkovnega tipa. Primer: Pri deklaracijah

```
type t1 = integer; t2 = t1; t3 = t1;
var  i1 : integer; i2 : integer;
     i3 : t1;
     i4 : t1; i5 : t2; i6 : t3;
     p1 : ^integer; p2, p3 : ^integer;
```

se tipsko prilagodita i1 in i2, i3 in i4 ter p2 in p3.

Primer: Podatkovni tipi

```
type a = b; b = c; c = ^a;
```

so veljavni.

Deklaracije

Vrstni red deklaracij podatkovnih tipov ni pomemben — pri deklaraciji podatkovnega tipa lahko uporabimo podatkovni tip, ki je definiran pred ali za deklaracijo tega tipa.

Vrstni red deklaracij podprogramov ni pomemben — pri deklaraciji podprograma lahko uporabimo podprogram, ki je definiran pred ali za deklaracijo tega podprograma.

Izrazi

Unarni operator `not` deluje nad vsakim podizrazom, katerega tip je strukturno enak tipu `boolean`. Tip celega izraza je enak tipu podizraza.

Unarna operatorja + in - delujeta nad vsakim podizrazom, katerega tip je strukturno enak tipu `integer`. Tip celega izraza je enak tipu podizraza.

Binarna logična operatorja delujeta nad podizrazoma, ki se lahko tipsko prilagodita in katerih tipa sta strukturno enaka tipu `boolean`. Tip celega izraza je enak tipu tistega podizraza, ki ni konstanten izraz.

Binarni primerjalni operatorji delujejo nad podizrazoma, ki se lahko tipsko prilagodita in katerih tipa sta strukturno enaka kazalca ali pa sta enaka enemu izmed tipov `integer`, `char`, `string` in `boolean`. Tip celega izraza je tip `boolean`.

Binarni aritmetični operatorji delujejo nad podizrazoma, ki se lahko tipsko prilagodita in katerih tipa sta strukturno enaka tipu `integer`. Tip celega izraza je enak tipu tistega podizraza, ki ni konstanten izraz.

Stavki

Na levi strani prireditvenega stavka mora biti izraz, ki je strukturno enak kazalcu ali enemu izmed tipov `integer`, `char`, `string` in `boolean`. Leva in desna stran prireditvenega stavka morata biti tipsko prilagodljivi.

V stavkih `if` in `while` mora biti tip pogoja strukturno enak tipu `boolean`. Spremenljivka v stavku `for` mora biti tipa, ki je strukturno enak tipu `integer`, izraza za spodnjo in zgornjo mejo zanke pa morata biti tipsko prilagodljiva tipu spremenljivke.

Podprogrami

Parametri podprogramov in rezultati funkcij morajo biti strukturno enaki kazalcu ali enemu izmed tipov `integer`, `char`, `string` in `boolean`.

Pri klicu podprograma se morajo vsi argumenti klica tipsko prilagoditi istoležnemu parametru podprograma.

Ostalo

Vsa ostala pravila so enaka kot pri programskem jeziku Pascal.

Slike

2.1 Internetni oblak 6
2.2 Plasti računalništva v oblaku 11

3.1 Zoho Writer 16
3.2 Aviary Phoenix 17
3.3 Noteflight 18
3.4 Google Maps 19
3.5 Mozilla Bespin 20
3.6 Salesforce 21

4.1 Vzorci za pretvorbo dreves vmesne kode v MMIX zbirnik 44
4.2 Nadzorna plošča Google App Engine. 55

Izvorna koda

4.1 pascal.lex . . . 33
4.2 pascal.cup . . . 34
4.3 Main.java (leksikalna in sintaksna analiza) . . . 34
4.4 SynTree.java . . . 35
4.5 SynProgram.java . . . 36
4.6 SynForStmt.java . . . 37
4.7 SynTypeDecl.java . . . 37
4.8 Main.java (semantična analiza) . . . 38
4.9 SynStringType.java . . . 38
4.10 SynWhileStmt.java . . . 38
4.11 SynUnExpr.java . . . 39
4.12 Main.java (generiranje vmesne kode) . . . 39
4.13 SynNilConst.java . . . 40
4.14 SynBinExpr.java . . . 40
4.15 SynFuncCall.java . . . 40
4.16 Main.java (generiranje ciljne kode) . . . 41
4.17 VarChunk.java . . . 42
4.18 ConstChunk.java . . . 42
4.19 CodeChunk.java . . . 43
4.20 code.mms . . . 44
4.21 compiler.jsp . . . 45
4.22 header.jsp . . . 46
4.23 web.xml . . . 46
4.24 appengine-web.xml . . . 47
4.25 CompilerServlet.java (prvi del) . . . 48
4.26 HtmlWriter.java . . . 49
4.27 CompilerServlet.java (drugi del) . . . 51
4.28 CompilerServlet.java (tretji del) . . . 52
4.29 SaveFileServlet.java . . . 52
4.30 cron.xml . . . 55

Literatura

[1] A. V. Aho, R. Sethi, J. D. Ullman,
"Compilers - Principles, Techniques and Tools", Addison-Wesley, 1986.

[2] A. W. Appel, "Modern Compiler Implementation in Java (2nd edition)", Cambridge University Press, 2002.

[3] D. Chappell, "A Short Introduction to Cloud Platforms", 2008

[4] (2009) S. Johnston, "The Cloud and Cloud Computing Consensus Definition?", Julij 2008. Dostopno na:
http://samj.net/2008/07/cloud-and-cloud-computing-consensus.html

[5] D. E. Knuth, "The Art of Computer Programming", zvezek 1 (MMIX), Addison-Wesley, 1999

[6] M. Sheehan, "The Past, Present and Future of The Cloud",
SOAWorld Magazine, December 2008.

[7] J. Varia, "Cloud Architectures". Dostopno na:
http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf

[8] S. Wardley, "Cloud Computing - Why IT Matters",
predavanje na konferenci *Open Source Convention*, Julij 2009.

[9] (2009) Spletna enciklopedija *Wikipedia*, gesla:
"Cloud Computing", "Software as a service", "Platform as a service", "Infrastructure as a service", "Cloud storage", "Compiler" in druga.
Dostopno na: http://en.wikipedia.org/

[10] (2009) "Cloud Computing Frequently Asked Questions".
Dostopno na: http://www.oracle.com/technologies/cloud/faq.html

[11] (2009) "Glossary of Dedicated Server Hosting Terms".
Dostopno na: http://www.servepath.com/support/definitions.php

[12] (2009) “Navigating the Layers of the Cloud Computing Pyramid”. Dostopno na: http://blog.gogrid.com/

[13] (2009) “Wekti Glossary”. Dostopno na: http://wekti.com/glossary/

[14] Dokumentacija Google App Engine. Dostopno na: http://code.google.com/appengine/docs/

[15] Spletne strani ponudnikov računalništva v oblaku.