Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jernej Demojzes

Tehnologija za masovni zajem in analizo podatkov iz družbenih omrežij ter njihova vizualizacija v realnem času

diplomsko delo

visokošolski strokovni študijski program
računalništvo in informatika

Mentor: viš. pred. dr. Alenka Kavčič

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00544 / 2013
Datum: 15.9.2013

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

Kandidat: **JERNEJ DEMOJZES**

Naslov: **TEHNOLOGIJA ZA MASOVNI ZAJEM IN ANALIZO PODATKOV IZ DRUŽBENIH OMREŽIJ TER NJIHOVA VIZUALIZACIJA V REALNEM ČASU**
TECHNOLOGY FOR HARVESTING, ANALYSIS AND REAL TIME VISUALIZATION OF DATA FROM SOCIAL NETWORKS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Družbena omrežja, kot sta Facebook in Twitter, neprestano ustvarjajo ogromne količine podatkov, ki pa se ob družbeno odmevnih dogodkih še povečajo. Analiza teh podatkov bi bila zelo zanimiva in uporabna tudi kot kazalnik mnenj o aktualnih dogodkih. V okviru diplomske naloge izdelajte porazdeljeno aplikacijo za pridobivanje velikih količin podatkov iz različnih družbenih omrežij ter njihovo analizo. Aplikacija naj deluje v oblaku, kar omogoča boljše prilagajanje glede na potrebne vire ter s tem tudi pridobivanje in obdelavo pridobljenih podatkov v realnem času. Implementirajte tudi spletno aplikacijo za vizualizacijo analiziranih podatkov. Na koncu podajte nekaj primerov uporabe in kritično ocenite delovanje aplikacije.

Mentor:
viš. pred. dr. Alenka Kavčič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jernej Demojzes, z vpisno številko **63060062**, sem avtor diplomskega dela z naslovom:

Tehnologija za masovni zajem in analizo podatkov iz družbenih omrežij ter njihova vizualizacija v realnem času

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Alenka Kavčič,
- 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 15. septembra 2013 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

API (angl. application programming interface) programski vmesnik

IAAS (angl. infrastructure as a service) infrastruktura kot storitev

PAAS (angl. platform as a service) platforma kot storitev

SAAS (angl. software as a service) aplikacija kot storitev

.NET Microsoftovo programsko ogrodje

PHP (angl. hypertext preprocessor) skriptni programski jezik za izdelavo spletnih strani

SQL (angl. structured query language) strukturiran povpraševalni jezik za delo s podatkovnimi bazami

IIS (angl. internet information services) Microsoftov spletni strežnik

SSMS (angl. Microsoft SQL server managment studio) Microsoftovo orodje za delo z podatkovnimi bazami

WCF (angl. Windows communications foundation) Microsoftovo ogrodje za izdelavo storitev

HTTP (angl. hypertext transfer protocol) protokol za izmenjavo hiperteksta ter grafičnih, zvočnih in drugih večpredstavnostnih vsebin na spletu

ADO.NET (angl. active data objects) Microsoftovo ogrodje za dostop do podatkovnih storitev

ASP.NET (angl. active server pages) Microsoftovo ogrodje za izdelavo spletnih strani

MVC (angl. model view controller) programski arhitekturni vzorec model-pogled-nadzornik

ODBC (angl. Open DataBase Connectivity) standardna SQL-metoda pristopa iz poljubne aplikacije do podatkovnih baz

IT (angl. information technology) informacijska tehnologija

CLI (angl. common language infrastructure, krat. CLI) skupna jezikovna infrastruktura

CLR (angl. common language runtime) okolje, ki poganja programe .NET

LINQ (angl. language-integrated query) sintaktična razširitev jezikov C-Sharp in Visual Basic

HTML (angl. hypertext markup language) označevalni jezik za oblikovanje večpredstavnostnih dokumentov

CSS (angl. cascading style sheet) stilna predloga na spletni strani, v kateri je zapisana oblika spletne strani

XML (angl. extensible markup language) format podatkov za izmenjavo strukturiranih dokumentov v spletu

SVG (angl. scalable vector graphics) označevalni jezik za opis dvorazsežne statične in risane vektorske grafike

CPU (angl. central processing unit) centralna procesna enota

GUID (angl. globally unique identifier) unikatna številka, ki se jo uporablja kot identifikator

GUI (angl. graphical user interface) grafični uporabniški vmesnik

JSON (angl. JavaScript object notation) format za prenos podatkov v obliki parov atribut-vrednost

URL (angl. uniform resource locator) internetni naslov, na katerem se nahaja vsebina

Povzetek

V diplomski nalogi je predstavljen razvoj aplikacije v oblaku, ki je sposobna ujeti veliko število sporočil in jih analizirati ter vizualizirati v realnem času.

Zaradi velike količine podatkov v družbenih omrežjih in še povečane količine le-teh ob socialno odmevnih dogodkih je nastala ideja o pridobivanju in analizi teh podatkov. Želeli smo ustvariti rešitev, ki je sposobna zajeti sporočila ob večjih dogodkih in nad njimi izvesti več vrst analiz, kot so število zadetkov, njihov doseg, besedni oblaki, sentiment, jezik ...

V diplomski nalogi sem predstavil koncepte in porazdeljene arhitekture razvoja v oblaku, ki so nas pripeljale do tega, da analizo in prikaz pridobljenih podatkov lahko izvedemo v skoraj realnem času.

Ključne besede:

Družabna omrežja, oblak, Windows Azure, analiza podatkov, programsko ogrodje .NET, porazdeljeno procesiranje, skaliranje

Abstract

In this thesis I describe the development process of an application that is able to capture, analyze and visualize a large number of messages in real time.

Due to large amount of data on social networks and an even larger amount of data during media-exposed events an idea was born to gather and analyze that data. We wanted to create a solution that was able to gather messages during large events and create numerous analysis based on that messages like hit count analysis, reach count analysis, word cloud analysis, sentiment count analysis, language analysis,...

In this thesis I describe the concepts of distributed architectural patterns in developing applications for the cloud that brought us to a point where we can do all of the above in near real-time.

Key words:

Social networks, cloud, Windows Azure, data analysis, .NET framework, parallel processing, scaling

Poglavje 1

Uvod

Družbena omrežja niso več le domena srednješolcev, temveč so postala vpliven medij sodobne družbe. Pričeli so jih uporabljati vplivni politiki v volilnih kampanijah, svetovni mediji, kot sta BBC in CNN, za širjenje novic med čim večje število bralcev, velike koorporacije, kot sta CocaCola in BMW, za komunikacijo z obstoječimi in potencialnimi strankami ter drugi. Pretok informacij v družbenih omrežjih je iz meseca v mesec večji. Ob ponovni izvolitvi ameriškega predsednika Baraca Obame je bilo v družbeno omrežje Twitter poslanih preko 327.000 sporočil na minuto, njegov tvit "Four more years" pa je bil do danes deljen s strani 340.000 uporabnikov.

Družbena omrežja so tako zaradi velike količine uporabnih podatkov postala zanimiva za analitike, a je zajem in analiza takšne količine podatkov pretrd oreh za običajne strežnike in klasične koncepte programiranja.

Cilj te diplomske naloge je predstaviti koncepte razvijanja porazdeljene aplikacije v oblaku, ki je sposobna zajeti čim večje število rezultatov, nad njimi izvesti različne analize in jih v končni fazi prenesti do končnega uporabnika, bodisi preko spletnega vmesnika, bodisi preko aplikacijsko programskega vmesnika (API-ja).

Aplikacija za analizo in vizualizacijo podatkov, ki je predstavljena v diplomski nalogi, je sestavljena iz treh modulov. Prvi modul ima nalogo zajema podatkov iz družbenih omrežij, drugi modul ima nalogo izvajanja vnaprej izračunanih analiz, tretji modul pa je namenjen spletnemu vmesniku. V tej nalogi bom predstavil koncepte, tehnologijo in arhitekturo, ki je bila uporabljena pri izdelavi

aplikacije in opisal njeno izdelavo.

1.1 Organizacija diplomske naloge

V drugem poglavju so opisane vse tehnologije, uporabljene pri izdelavi aplikacije, od oblaka do spletnih knjižnic, uporabljenih za risanje grafov.

Tretje poglavje govori o razvoju rešitev v oblaku z uporabo porazdeljenega procesiranja, porazdeljevanjem bremena, upravljanja z bremenom in uporabo federiranih podatkovnih baz.

V četrtem poglavju je predstavljen koncept izdelave aplikacije. Natančno je opisana arhitektura aplikacije in koncepti, ki so uporabljeni pri izdelavi modula za pridobivanje podatkov, modula za računanje analiz ter modula za spletni vmesnik.

Poglavje 2

Uporabljene tehnologije

2.1 Oblak

Oblak ali bolje rečeno računalništvo v oblaku je po definiciji [2] izraz, ki opisuje različne tipe računalniških konceptov in modelov, ki omogočajo enostaven oddaljen dostop do velikega števila računalniških virov, ki jih je mogoče po potrebi povečati ali zmanjšati brez fizične interakcije s samimi strežniki.

Za oblak veljajo naslednje karakteristike [2]:

- Enostavno skaliranje števila in velikosti virov brez človeške interakcije (število virtualnih strežnikov, velikost baze, procesorska moč ...).
- Širokopasovni internetni dostop do virov v oblaku.
- Upravljanje z viri, ki omogoča porazdeljeno uporabo več fizičnih in virtualnih virov, ki se lahko nahajajo na različnih kontinentih. Število virov se lahko dinamično povečuje glede na potrebe, pri tem pa na zunaj ni občutka porazdeljenosti.
- Elastičnost virov, ki omogoča enostavno skaliranje kapacitet, včasih tudi samodejno.
- Nadzor nad številom, velikostjo in zasedenostjo virov.

2.1.1 Namestitveni modeli oblakov

Obstajajo štirje različni namestitveni modeli oblaka [3]: javni, zasebni, hibridni in skupnostni.

2.1.1.1 Javni oblak

Pri javnem oblaku ponudnik storitev v oblaku skrbi za infrastrukturo in storitve, do katerih ima dostop večje število klientov.

Ta model oblaka je primeren za uporabnike, ki želijo zagotoviti stabilnost aplikacije pri povečanju prometa v konicah in se izogniti stroškom nameščanmja lastne infrastrukture.

Ponudniki javnih oblakov so: Amazon, Google, Microsoft ...

2.1.1.2 Zasebni oblak

Zasebni oblak je primeren za uporabnike, ki jim je varnost na prvem mestu. Postavitev oblaka in njegovo vzdrževanje se izvaja za vsako stranko posebej, kar privede do veliko večjih stroškov kot pri javnem oblaku.

Varnost v zasebnih oblakih se zagotavlja z virtualnimi zasebnimi omrežji ali s fizično postavitvijo infrastrukture znotraj požarnih zidov.

Postavitev oblaka je znotraj podjetja ali na oddaljeni lokaciji. Če želimo do podatkov dostopati tudi ob izpadu internetnega omrežja, moramo oblak namestiti znotraj podjetja.

2.1.1.3 Hibridni oblak

Hibridni oblak shranjuje varovane podatke in aplikacije v zasebnem oblaku, javno dostopne podatke in aplikacije pa v javnem oblaku. Zasebni oblak si lahko pomaga z javnim tudi ob visokih zahtevkih za procesiranje ali močno povečanem prometu.

2.1.1.4 Skupnostni oblak

V skupnostnem oblaku je infrastruktura deljena med več organizacij, ki si delijo iste zahteve po varnosti. Ta model omogoča zmanjšanje stroškov, ki bi jih prinesla namestitev lastnega zasebnega oblaka, obenem pa poskrbi za enako varnost podatkov kot zasebni oblak.

Ta model je pogosto uporabljen s strani različnih vladnih organizacij.

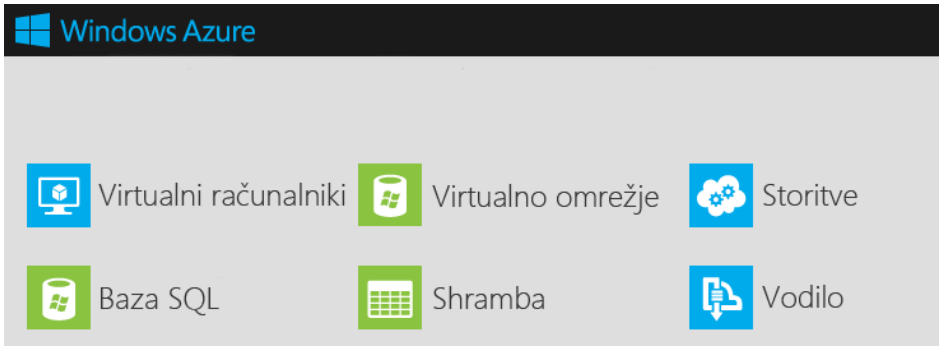
2.1.2 Storitve v oblaku

Spekter ponudnikov storitev v oblaku kot tudi spekter samih storitev, ki jih ti ponujajo, je iz dneva v dan večji. Glede na tip storitev delimo oblake v tri skupine IaaS, PaaS in SaaS [4].

2.1.2.1 Infrastruktura kot storitev

Infrastruktura kot storitev (angl. Infrastructure as a Service) ali model IaaS omogoča uporabo infrastrukturnih virov v oblaku. Kot take štejemo: virtualne računalnike, virtualna omrežja, skladiščenje podatkov ... Primer infrastrukturnih storitev v oblaku Windows Azure je prikazan na sliki 2.1. Pri takšnem modelu uporabnik nase prevzame nalogo posodabljanja programske opreme in operacijskega sistema.

Največja prednost modela IaaS je, da uporabniku omogoča nadzor nad komponentami, ki jih želi uporabiti. Število komponent, njihovo velikost in hitrost je mogoče spreminjati na daljavo brez fizične interakcije, s samo strojno opremo, ali interakcije z vzdrževalci sistema.



Slika 2.1: Vrste infrastrukturnih storitev v oblaku Windows Azure [5]

2.1.2.2 Platforma kot storitev

Platforma kot storitev (angl. Platform as a Service) ali model PaaS nudi platformo kot storitev v obliki operacijskega sistema s prednaloženimi aplikacijami, ki jo uporabnik nato uporabi kot sistem, na katerem se izvaja njegova aplikacija.

Največja prednost modela PaaS je, da se uporabniku ni potrebno obremenjevati z namestitvijo ter posodobitvami operacijskega sistema in vseh ostalih

aplikacij, potrebnih za delo.

2.1.2.3 Aplikacija kot storitev

Aplikacija kot storitev (angl. Software as a Service) ali model SaaS ponuja aplikacije kot storitve v oblaku, dostop do njih pa je mogoč preko različnih naprav, kot so mobilni telefoni, računalniki in tablice.

Največja prednost tega modela je, da ni potreben finančni vložek v strežnike in programske licence.

Primeri modelov oblakov SaaS so Google Drive, Office 365 in Dropbox.

2.1.3 Windows Azure

Windows Azure je Microsoftova platforma javnega oblaka, ki omogoča enostaven razvoj, namestitev in upravljanje z aplikacijami na globalni mreži podatkovnih centrov [5].

Omogoča razvoj aplikacij v poljubnem jeziku, ogrodju in urejevalniku. Med drugim podpira programske platforme .NET, PHP, Javo, Node.js, Python, Ruby ...

Nadgradnje operacijskega sistema in servisov se izvajajo samodejno. Platforma podpira tudi samodejno upravljanje z bremenom in zagotavlja 99,9% mesečno razpoložljivost.

Vire je mogoče enostavno skalirati glede na potrebe v danem trenutku, kar omogoča bistveno zmanjšanje stroškov.

V platformi Windows Azure imamo dostop do številnih storitev, ki so razdeljene v štiri večje sklope:

- Računske storitve (virtualni računalniki, spletne strani, mobilne storitve in storitve oblaka).
- Podatkovne storitve (Windows Azure baza SQL, storitev blob ...).
- Aplikacijske storitve (aktivne mape, medijske storitve ...).
- Omrežne storitve (virtualna omrežja in upravljalec prometa).

V tej diplomski nalogi so uporabljene storitve Azure Cloud Services in Windows Azure SQL database.

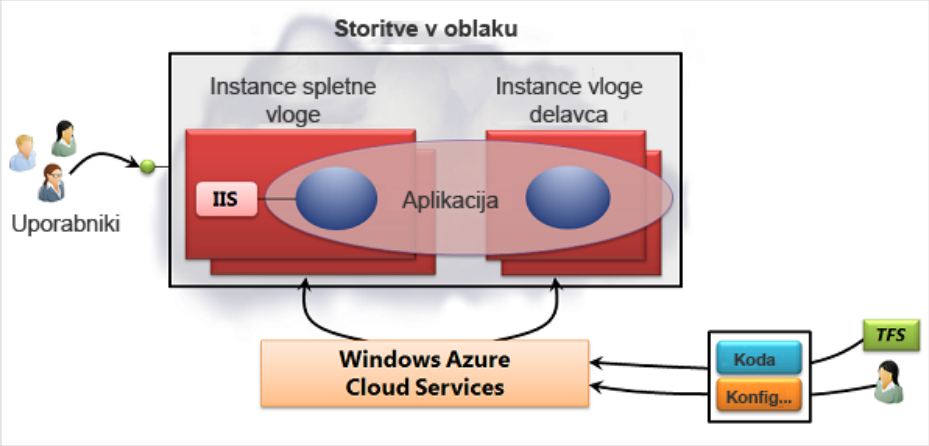
2.1.3.1 Azure Cloud Services

Azure Cloud Services zagotavljajo platformo kot storitev, kar je idealno za aplikacijo za analizo in vizualizacijo podatkov, saj lahko število instanc (virtualnih strežnikov v infrastrukturi oblaka, ki služijo kot primerek vloge) poljubno povečamo, kar nam zagotovlja hitrejše delovanje ob spremljanju večjih dogodkov, ali poljubno zmanjšamo, ko je prometa malo in tako zmanjšamo stroške. Prav tako se nam ni potrebno ukvarjati z infrastrukturo, kar prihrani razvijalcem kar precej časa.

Na voljo imamo dve različni vrsti vlog: spletno vlogo (angl. web role) in vlogo delavca (angl. worker role). Obe vrsti vlog v ozadju tečeta na virtualnih računalnikih. Spletna vloga teče na strežniku Windows z nameščenim IIS, pri čemer vloga delavca teče na strežniku Windows brez IIS (slika 2.2).

Do vseh virtualnih računalnikov dostopamo preko istega naslova IP, storitev bo pa namestila virtualne računalnike tako, da ne bodo podvrženi fizičnim okvaram. Upravljalnik omrežnega bremena poskrbi, da se zahtevki enakomerno porazdelijo med virtualne računalnike.

V primeru, da se eden od računalikov, na katerem teče naša aplikacija, pokvari, bo storitev to zaznala in pognala virtualni računalnik na drugem strežniku. Prav tako zazna prenehanje delovanja aplikacije zaradi programske napake in aplikacijo ponovno zažene.



Slika 2.2: Windows Azure Cloud Services [5]

2.1.3.2 Windows Azure SQL Database

Poleg ostalih storitev za shranjevanje podatkov, kot so Blob in virtualni računalniki z strežnikom SQL, nam Windows Azure nudi tudi relacijsko podatkovno bazo, ki nam z oblakom prinaša številne prednosti, kot so enostavno skaliranje in visoka stopnja dostopnosti in zanesljivosti.

Za razliko od virtualnih računalnikov z naloženim strežnikom SQL tukaj plačamo le prostor, ki ga uporabljamo. Velikost baze lahko po potrebi povečujemo do 150 GB. Če nam je to premalo, lahko uporabimo federirane podatkovne baze in na podlagi ene ali več tabel razdelimo bazo na več manjših federacij in tako pospešimo delovanje aplikacije, ter si zagotovimo večji prostor za shranjevanje podatkov.

2.1.3.2.1 Arhitektura

Windows Azure SQL database teče na strežnikih SQL v Microsoftovih podatkovnih centrih. Kot je razvidno s slike 2.3, so štirje različni nivoji arhitekture, ki zagotavljajo delovanje podatkovne baze: nivo odjemalca, nivo storitev, nivo platforme in nivo infrastrukture.

Nivo odjemalca

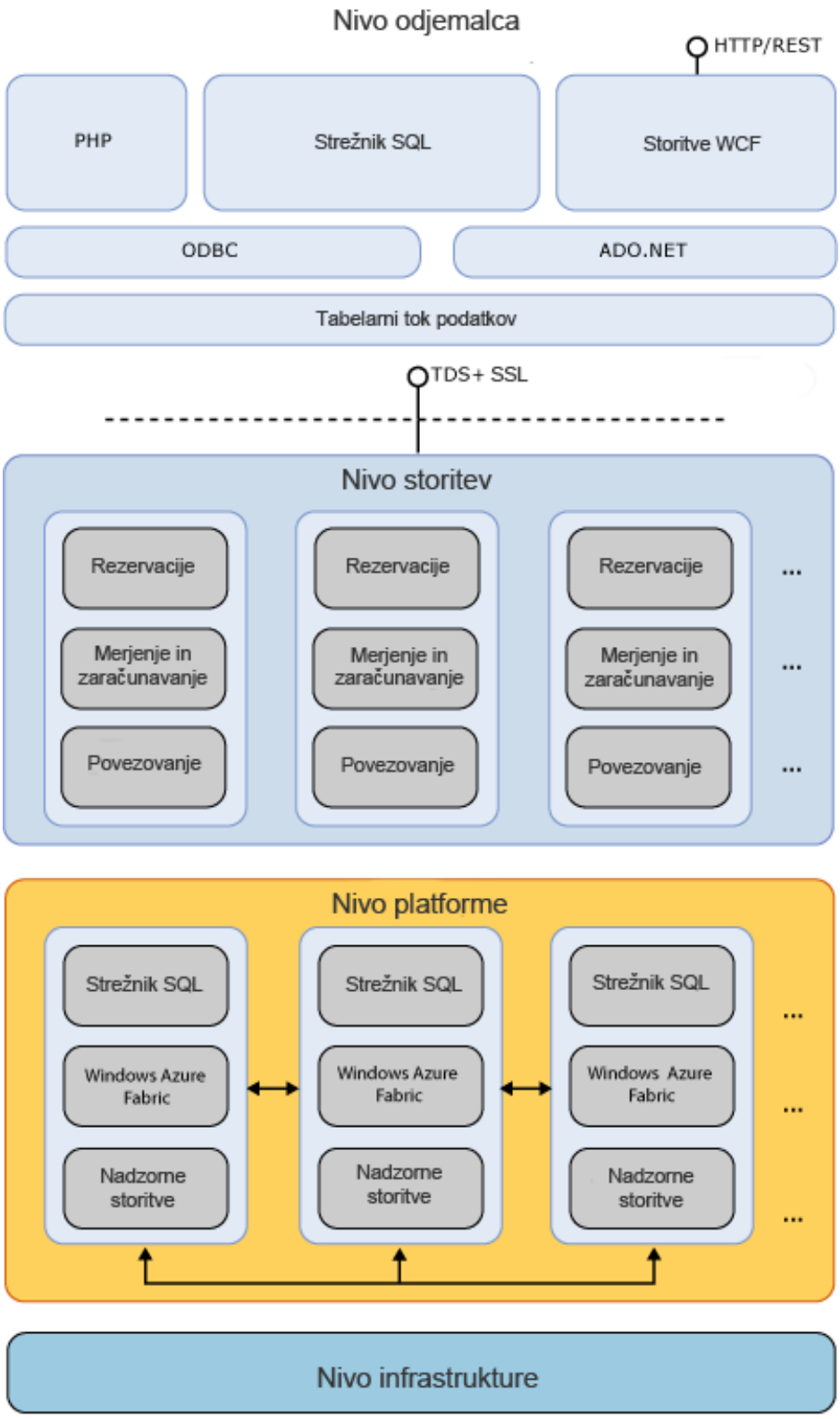
Nivo odjemalca omogoča aplikaciji dostop do podatkovne baze. Nahaja se lahko v oblaku ali na lastnih strežnikih. Za prenos podatkov uporablja isti tabelarični podatkovni tok kot navaden strežnik Microsoft SQL Server, kar nam omogoča uporabo orodij, kot je Microsoft SQL Server Management Studio (SSMS).

Nivo storitev

Nivo storitev skrbi za povezavo med nivojem odjemalca in nivojem platforme tako, da preusmerja klice od odjemalca na enega izmed mnogih strežnikov, na katerih teče podatkovna baza. Zagotavlja tudi nadzor nad prometom in računanje stroškov.

Nivo platforme

Nivo platforme je sestavljen iz večih strežnikov SQL. Omogoča samodejno okrevanje, razporejanje bremena in replikacijo strežnikov.



Slika 2.3: Arhitektura Windows Azure SQL Database [5]

Nivo infrastrukture
Nivo infrastrukture predstavlja IT administracijo strojne opreme in operacijskega sistema.

2.2 Ogrodje .NET

Ogrodje .NET je razvil Microsoft za razvijanje in poganjanje aplikacij na operacijskih sistemih Windows [6].

Storitve, ki jih ogrodje .NET zagotavlja aplikacijam, so naslednje:

- Upravljanje s pomnilnikom.
- Skupni sistem tipov, ki je definiran v ogrodju .NET in omogoča uporabo istih tipov v različnih jezikih.
- Zbirke knjižnic razredov, ki uporabnikom olajšujejo pisanje nizkonivojske kode.
- Razvijalska ogrodja in tehnologije, kot so ASP.NET, ADO.NET ...
- Zagotavljanje kompatibilnosti med programskimi jeziki tako, da se vsi jeziki pretvorijo v vmesno kodo (CLI), preden se prevedejo in poženejo v izvajalnem okolju (CLR).
- Kompatibilnost med verzijami.
- Hkratno poganjanje večih verzij ogrodja in aplikacij.
- Kompatibilnost aplikacij z operacijskimi sistemi, kot so Windows 7, Windows 8, Windows Phone in XBox 360.

Ogrodje .NET je sestavljeno iz dveh delov: izvajalnega okolja CLR (angl. Common Language Runtime) in zbirke knjižnic razredov (angl. extensive class library).

2.2.1 Izvajalno okolje CLR

Izvajalno okolje CLR opravlja naslednje funkcije:

- Upravljanje s pomnilnikom.

- Izvajanje niti.
- Izvajanje kode.
- Preverjanje varnosti kode.
- Prevajanje kode.
- Ostale sistemske funkcije.

2.2.2 Zbirka knjižnic razredov

Zbirka knjižnic razredov vsebuje tipe, ki so lahko uporabljeni v vseh programskih jezikih, ki so del ogrodja .NET. Te lahko naprej dedujemo, kar nam zelo olajša delo.

Zbirka knjižnic razredov nam omogoča:

- Delo z nizi.
- Uporabo podatkovnih struktur.
- Povezavo s podatkovno bazo.
- Dostop do datotek.
- Razvoj konzolnih aplikacij.
- Razvoj Windows aplikacij.
- Razvoj spletnih aplikacij.

2.3 C-Sharp

C# je programski jezik, ki ga je razvil Microsoft, za razvoj aplikacij na ogrodju .NET. Jezik je že na prvi pogled podoben jezikom C, C++ in Java, kar omogoča večini razvijalcev lahek prehod.

C# prinaša funkcionalnosti, kot so nulabilni tipi, enumeracije, delegate, lambda izraze in neposreden dostop do pomnilnika. Sintaktična razširitev LINQ (angl. Language-Integrated Query), ki je bila vpeljana z ogrodjem .NET 3.5, omogoča tudi pisanje poizvedb, ki so po sintaksi in uporabnosti podobne poizvedbam SQL.

2.4 Spletne tehnologije

Razvoj uporabniškega vmesnika na spletu prinaša za uporabnike številne ugodnosti, saj niso več odvisni od odjemalske platforme, za poganjanje aplikacije pa ni potrebno predhodno nameščati nobene programske opreme.

2.4.1 HTML in CSS

Označevalni jezik HTML (angl. Hypertext Markup Language) je jezik za strukturiranje spletnih strani. Dokument sestavimo iz elementov HTML, kot so , <h1>, <p> ... Vsak element ima praviloma zaključni element, ki se prične s poševnico ( ). Elementi imajo lahko atribute ( ), znotraj elementa pa lahko vstavimo druge elemente in tekst. Zadnja verzija jezika je HTML5, ki prinaša veliko novosti, kot so video elementi, vektorska grafika ...

Za stiliranje dokumentov HTML se uporablja jezik CSS. Ta podpira nastavitve barv, pisave, velikosti ... Jezik CSS je neodvisen od jezika HTML in ga lahko uporabljamo s katerimkoli jezikom XML [7].

2.4.2 SVG

Jezik SVG (angl. Scalable Vector Graphics) se uporablja za opis slik z vektorskimi elementi, tekstom in rastorsko grafiko. SVG lahko uporabljamo znotraj dokumenta HTML, podpirajo ga pa vsi moderni brskalniki.

Sliko SVG sestavimo podobno kot dokument HTML z elementi, ki imajo atribute, znotraj njih pa lahko gnezdimo druge elemente. Tako kot HTML tudi SVG podpira CSS stiliranje [8].

Nekaj primerov elementov SVG: <circle>, <elipse>, <path>, <line>, <rect>, <animate> ...

2.4.3 JavaScript

JavaScript je skriptni objektni programski jezik, zasnovan za potrebe interakcije z dokumentom HTML. Podpirajo ga praktično vsi brskalniki, vendar ga najdemo tudi v drugih okoljih, kot je Node.js [9].

JavaScript je široko razširjen jezik in ima posledično veliko število odprtokodnih knjižnic, ki nam olajšajo delo.

2.4.3.1 JQuery

JQuery je brez dvoma najbolj razširjena knjižnica JavaScript. Vsebuje širok spekter ukazov, ki nam olajšajo manipuliranje elementov HTML, olajšajo delo z tabelami in poenostavijo klice Ajax [10].

Uporabo knjižnice JQuery omogočajo vsi večji brskalniki.

2.4.3.2 D3.js

D3.js je knjižnica JavaScript za manipulacijo z dokumenti na podlagi podatkov. Podatke lahko vizualiziramo s pomočjo jezikov HTML, CSS ali SVG [11].

Vizualizacijo podatkov rešuje na precej drugačen način kot druge knjižnice za risanje grafov. Risanje grafov tako ni le pošiljanje pravilno formatirane tabele v funkcijo, ki nam izriše graf, temveč povezovanje tabel s podatki z elementi, ki nam jih pomagajo prikazati. Na ta način imamo veliko večjo kontrolo in lahko ustvarimo interaktivne grafe z animacijami med spremembami podatkov.

Poglavje 3

Razvoj rešitev v oblaku

V tem poglavju bomo predstavili značilnosti razvoja rešitev v oblaku in se osredotočili na posebnosti, ki jih moramo pri tem upoštevat. Kot prvo bomo podrobneje opisali skaliranje v oblaku, nato predstavili porazdeljevanje bremena, izborne algoritme in na koncu še federirane podatkovne baze.

3.1 Skaliranje v oblaku

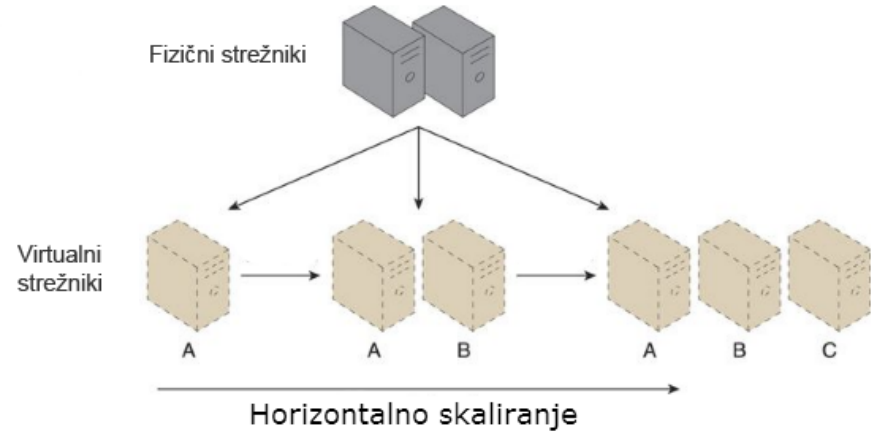
Skaliranje v oblaku pomeni, da se viri, ki jih plačujemo, prilagajajo porabi, ki jo ima naša aplikacija, tako da se bodisi poveča število instanc v konicah bodisi zmanjša, ko je obremenitev aplikacije manjša.

Moč in število instanc lahko nastavimo sami ali pa aplikacijo oziroma storitev skonfiguriramo tako, da to stori sama glede na obremenitev virov ali dolžino čakalne vrste. Temu pravimo elastično skaliranje.

Glede na način prilagajanja virov ločimo horizontalno in vertikalno skaliranje, ki sta podrobneje opisana v naslednjih poglavjih.

3.1.1 Horizontalno skaliranje

Dodajanje ali odstranjevanje instanc iste vrste je horizontalno skaliranje (slika 3.1). Dodajanje instanc je skaliranje navzven, odvzemanje instanc pa skaliranje navznoter.



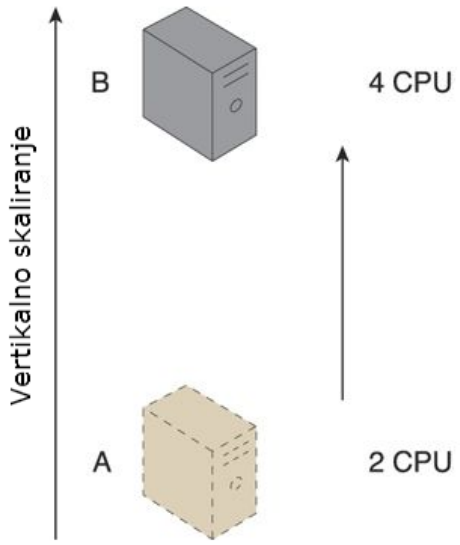
Slika 3.1: Horizontalno skaliranje z ene najprej na dve, nato še na tri instance [1]

Primeri, pri katerih je pametno uporabiti horizontalno skaliranje, so naslednji:

- Ko ima naša aplikacija ob konicah veliko višji promet kot ob normalnem delovanju.
- Ko ima uporabnike oziroma storitve porazdeljene na različnih lokacijah.
- Ko imamo zahtevke v vrsti in lahko vsaka instanca neodvisno procesira svoj delež.

3.1.2 Vertikalno skaliranje

Ko instanci povečamo ali zmanjšamo procesorsko moč, velikost pomnilnika ali katero izmed ostalih karakteristik oziroma jo zamenjamo z močnejšo instanco, temu pravimo vertikalno skaliranje (slika 3.2). Povečevanje kapacitete instance je skaliranje navzgor, zmanjšanje pa skaliranje navzdol.



Slika 3.2: Vertikalno skaliranje z dvoprocesorske instance na štiriprocesorsko [1]

Vertikalno skaliranje je v oblaku manj pogosto, zanj se odločimo, ko instance niso dovolj učinkovite z viri, ki jih imajo trenutno na voljo.

3.2 Porazdeljevanje bremena

Pri horizontalnem skaliranju se breme porazdeli na večje število instanc, da se poveča računsko zmogljivost ali spominsko kapaciteto virov. Mehanizem, ki skrbi za porazdeljevanje bremena med instancami, se imenuje upravljalnik bremena.

Upravljalnik bremena lahko opravlja več distribucijskih funkcij, med katerimi so:

- **Simetrična distribucija bremena** porazdeli delo simetrično med instance, kar je smiselno uporabiti takrat, ko je procesiranje vsakega izmed sporočil enako, oziroma takrat, ko približno enako obremeni instanco.
- **Asimetrična distribucija bremena** porazdeli delo tako, da instanca z večjo procesorsko močjo dobi delo, ki je bolj zahtevno za procesiranje.

- **Prioritetna distribucija bremena** porazdeli delo tako, da pri tem upošteva prioriteto sporočila.
- **Vsebinsko zavedna distribucija bremena** porazdeli delo med instance glede na vsebino sporočila.

Upravljalnik bremena je sprogramiran oziroma skonfiguriran na podlagi zmogljivostnih pravil in parametrov z namenom optimizacije obremenitve instanc, preprečevanja njihove preobremenitve in zagotavljanja maksimalne pretočnosti. Ponavadi se nahaja med generatorjem bremena in med instancami, ki to delo opravljajo. Generator bremena je lahko instanca, ki generira delo, ali uporabniki, ki pošiljajo sporočila, ki jih je potrebno sprocesirati.

Upravljalnik bremena je lahko postavljen na naslednje načine:

- Kot večnivojsko omrežno stikalo.
- Kot strojna komponenta, namenjana porazdeljevanju vhodnih zahtevkov.
- Kot program, ki skrbi za porazdeljevanje bremena.
- Kot storitev, ki nam jo zagotavlja ponudnik oblaka.

3.3 Izborni algoritmi

Porazdeljeni sistemi v večini primerov potrebujejo vodjo, ki izvaja koordinacijske operacije, potrebne za optimalno delovanje preostalih instanc. Vsaka instanca mora vedeti, kdo je vodja, da lahko z njim komunicira. Problem nastane, ko vodilna instanca preneha delovati in je potrebno izbrati novo vodilno instanco. To se izbere s pomočjo izbornega algoritma.

Večina izbornih algoritmov temelji na naslednjih točkah:

- Vsaka instanca ima svojo prioritetno številko.
- Ob izboru postane instanca z najvišjo prioritetno številko vodja.
- Ob ponovnem zagonu instance se ta lahko pridruži preostalim delujočim instancam.

Najpogosteje se uporabljajo izborni algoritmi Bully, Ring in Sandipan Basu, ki so opisani v nadaljevanju [12].

3.3.1 Algoritem Bully

Pri tem algoritmu ima vsaka instanca prioritetno številko, seznam preostalih instanc in njihovih prioritetnih številk. Ko instanca pošlje zahtevek vodilni instanci in v določenem časovnem obdobju ne dobi odgovora, predvideva, da vodilna instanca ne deluje več in prične s postopkom izbora nove instance. Vsem instancam, ki imajo višjo prioritetno številko, pošlje izborno sporočilo. Če v določenem časovnem obdobju ne dobi odgovora, predvideva, da ima med aktivnimi instancami najvišjo prioritetno številko, in prevzame mesto vodilne instance. Takoj zatem pošlje sporočilo o novi vodilni instanci vsem instancam z nižjo prioritetno številko. Če pa dobi odgovor od katere izmed instanc z višjo prioritetno številko, počaka, da se postopek izbora izvede in dobi sporočilo o novi vodilni instanci.

Če instanca dobi izborno sporočilo od instance z nižjo prioritetno številko, ji odgovori, da je aktivna, in sproži postopek izbora med instancami z višjo prioritetno številko. Na koncu volitev je tako vedno izvoljena instanca z najvišjo prioritetno številko.

Instanca, ki se je po ustavitvi ponovno zagnala, mora sprožiti ponovni izbor in če ima ta instanca najvišjo prioritetno številko, postane vodilna instanca.

3.3.2 Algoritem Ring

Pri tem algoritmu so instance organizirane v obroč, po katerem se sporočila prenašajo vedno v isti smeri. Ko se sporočilo pošlje naprej, lahko instanca preskoči eno ali več instanc, če te niso aktivne.

Če instanca po poslanem sporočilu vodilni instanci ne dobi odgovora, predvideva, da ta ni več aktivna, in prične izborni postopek. Svojemu nasledniku pošlje izborno sporočilo, v katerem je prioritetna številka instance. Sporočilo se pošilje v krogu, pri čemer vsaka instanca doda svojo prioritetno številko. Ko sporočilo naredi celoten krog, ga instanca, ki ga je poslala, zazna kot svojega in iz seznama instanc izbere tisto, ki ima najvišjo prioritetno številko, in jo določi kot vodilno instanco.

Sporočilo z vodilno instanco pošlje v krogu, tako da je sprejeto s strani vseh instanc. Ko sporočilo naredi celoten krog, ga instanca, ki ga je kreirala, tudi uniči.

Instanca, ki se je po ustavitvi ponovno zagnala, pošlje sporočilo s svojo iden-

tifikacijo svojemu nasledniku. Sporočilo gre od instance do instance, dokler ne pride do vodilne instance, ki ostaja vodilna in to sporoči novo zagnani instanci.

Pri tem algoritmu ni nujno, da je vedno instanca z najvišjo prioritetno številko tudi vodilna instanca.

3.3.3 Algoritem Sandipan Basu

Pri tem algoritmu imajo vse instance svojo prioritetno številko, statusno tabelo s prioritetnimi številkami vseh instanc in so medsebojno povezane. Ko instanca pošlje sporočilo vodilni instanci in v določenem časovnem obdobju ne dobi odgovora, predvideva, da vodila instanca ni več aktivna, in sproži izborni postopek. Instanci z najvišjo prioritetno številko, ki ni trenutni vodja, pošlje izborno sporočilo.

Instanca, ki dobi izborno sporočilo, odgovori s potrditvenim sporočilom, nato pa pošlje sporočilo vsem instancam z nižjo prioritetno številko, da prevzema vlogo vodilne instance.

Če instanca ne dobi odgovora na izborno sporočilo v določenem časovnem obdobju, bo izbrala naslednjo najvišjo instanco in ji poslala izborno sporočilo. To ponavlja, dokler ne dobi odgovora ali ne pride po tabeli do svoje številke. V tem primeru pošlje vsem instancam z nižjo prioritetno številko, da prevzema vlogo vodilne instance.

Instanca, ki se po ustavitni ponovno zažene, pošlje zahtevek za statusno tabelo eni od svojih sosed. Nato preveri, če je njena prioritetna številka nižja od prioritetne številke vodilne instance. Če je prioritetna številka vodilne instance višja od njene, bo poslala sporočilo vsem instancam, da posodobijo statusno tabelo z novo aktivno instanco. Če pa je njena prioritetna številka višja od prioritetne številke trenutne vodilne instance, pošlje sporočilo vsem ostalim instancam, da prevzema vlogo vodilne instance.

3.4 Federirane podatkovne baze

Federirane podatkovne baze so način uporabe podatkovnih baz, ki nam s porazdelitvijo podatkov na več podatkovnih baz omogoča več prostora in hitrejše delovanje nad ključnimi tabelami SQL. Podatki so porazdeljeni po federacijah na podlagi ene ali več tabel. Uporaba federacij je smiselna, ko imamo v tabelah

takšno količino podatkov, da nam zmanjkuje prostora ali nam količina podatkov upočasnjuje delovanje aplikacije.

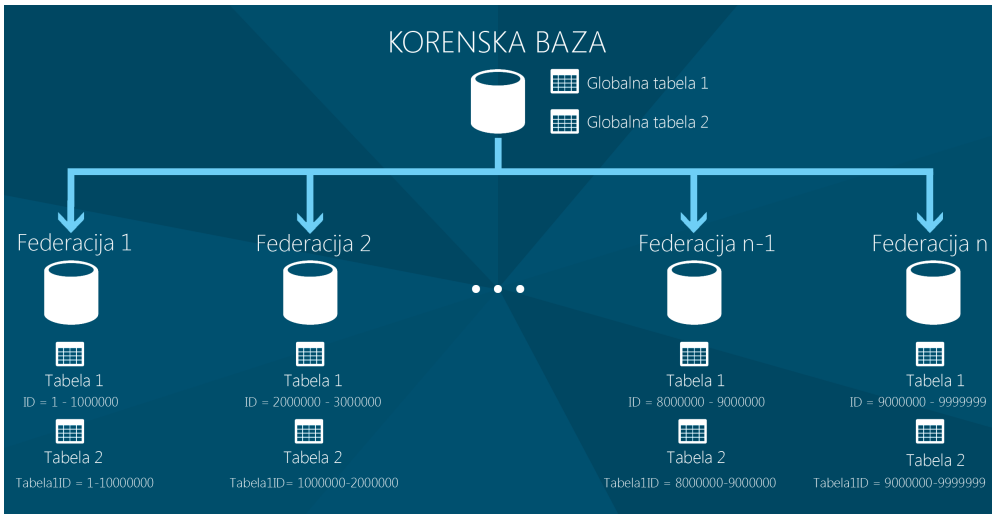
Število federacij je mogoče dinamično povečevati ali zmanjševati brez ustavljanja aplikacije ali nedostopnosti podatkov.

3.4.1 Arhitektura

Federacija je skupnost particij podatkovnih baz, ki jih definira federacijska shema. V federacijski shemi je definiran federacijski ključ porazdelitve, na podlagi katerega se nato podatki porazdelijo v eno izmed particij. Za vsako federacijo lahko obstaja samo ena shema in samo en federacijski ključ.

Vsaka particija v federaciji se imenuje federacijski član in vsebuje del podatkov s federacijskim ključem, ki ima vrednosti v območju, definiranem v federacijski shemi. Tabele, ki so porazdeljene po federacijah, imenujemo federacijske tabele.

Federacije so združene v korenski podatkovni bazi, ki lahko vsebuje eno ali več različnih federacij z različnimi federacijskimi shemami (slika 3.3). V korenski bazi so lahko globalne tabele s podatki o uporabnikih, pravicah ...



Slika 3.3: Arhitektura federiranih podatkovnih baz

Čeprav so federacije med seboj fizično ločene, se navzven obnašajo kot ena sama podatkovna baza. Do njih dostopamo preko istega naslova, odzivajo se

na isto ime, le ob klicu moramo podati vrednost ključa ali celotno območje vrednosti, da korenska baza ve, na katero federacijo nas mora povezati.

3.4.2 Delo z federacijami

Pri načrtovanju federacij je najbolj pomembna izbira prave tabele, na podlagi katere bomo porazdelili federacije. Izbrati moramo tako, da bomo dokaj enakomerno porazdelili tabele in do ključnih podatkov dostopali le iz ene federacije. V večini primerov je za porazdelitveno tabelo primerna tabela uporabnikov. Če pa ima nekaj uporabnikov veliko večje število zapisov od drugih, je pametno razmisliti o kakšni drugi izbiri.

Ključ porazdelitvene tabele moramo generirati tako, da se bodo podatki vsaj približno enakomerno porazdelili. Tu je nadvse priporočljiva uporaba unikatne identifikacijske številke GUID (podatkovni tip uniqueidentifier), ki jo lahko naključno generiramo že v aplikaciji. Če bi za ključ uporabili celo število (podatkovni tip bigint) in ga z vsakim novim zapisom povečali za 1, bi bili vsi zapisi v prvi federacji, dokler ne bi prešli območja prve federacije in začeli polniti drugo in tako naprej. Takšna porazdelitev za federacije v našem primeru ni primerna.

Če želimo pridobiti vse podatke iz ene izmed porazdeljenih tabel, uporabimo razpršene (angl. fan-out) poizvedbe. V praksi to izgleda tako, da se aplikacija poveže na vsako izmed federacij, izvede poizvedbo in podatke združi v eno tabelo, po možnosti vzporedno in ob istem času.

Poglavje 4

Izdelava aplikacije za analizo in vizualizacijo podatkov

Aplikacija za analizo in vizualizacijo podatkov (v nadeljevanju aplikacija) je inovativna rešitev, ki nam prinaša vpogled v razmišljanje širokega dela javnosti tako, da spremlja in analizira objave družbenih omrežij v skoraj realnem času (zaostanek petih sekund pri najvišji prioriteti). S kreiranjem iskalnih kriterijev (ključnih besed, lokacij, uporabnikov ...) ima uporabnik sam nadzor nad temo, ki jo aplikacija spremlja in zanjo ustvari analize, kot so analiza zadetkov, analiza sentimenta (rudarjenje mnenja, kjer objavi pripišemo pozitiven ali negativen odnos avtorja do objavljene vsebine), analiza dosega, besedni oblaki ...

V okviru diplomskega dela opisujem izdelavo aplikacije v oblaku, ki zajame velike količine podatkov iz družbenih omrežij v skoraj realnem času, nad njimi izvede vse zgoraj omenjene analize ter poskrbi za njihovo vizualizacijo na podlagi naprednih tehnologij za zajem, procesiranje, analizo in shranjevanje filtriranih sporočil iz računalniških sistemov. Platforma je neodvisna in jo je mogoče integrirati v druge sisteme IT. Storitev, ki se uporablja za analizo sentimenta rezultatov, je bila razvita na Institutu Jožef Stefan.

Namen rešitve je spremljanje javnega mnenja v realnem času. Glavni cilj rešitve pa je numerična analiza v realnem času in analiza vsebine na zahtevo v

čim krajšem času z namenom učenja in prilagajanja prihodnjih akcij.

4.1 Arhitektura aplikacije

Aplikacija teče v storitvi Windows Azure Cloud Services in je zasnovana tako, da izkorišča veliko število prednosti, ki nam jih ponuja razvoj v oblaku. Kot je razvidno s slike 4.1, je aplikacija sestavljena iz spletne vloge (Web.GUI), vloge črpalke (Worker.Pump), vloge procesorja (Worker.Processor) in federirane podatkovne baze.

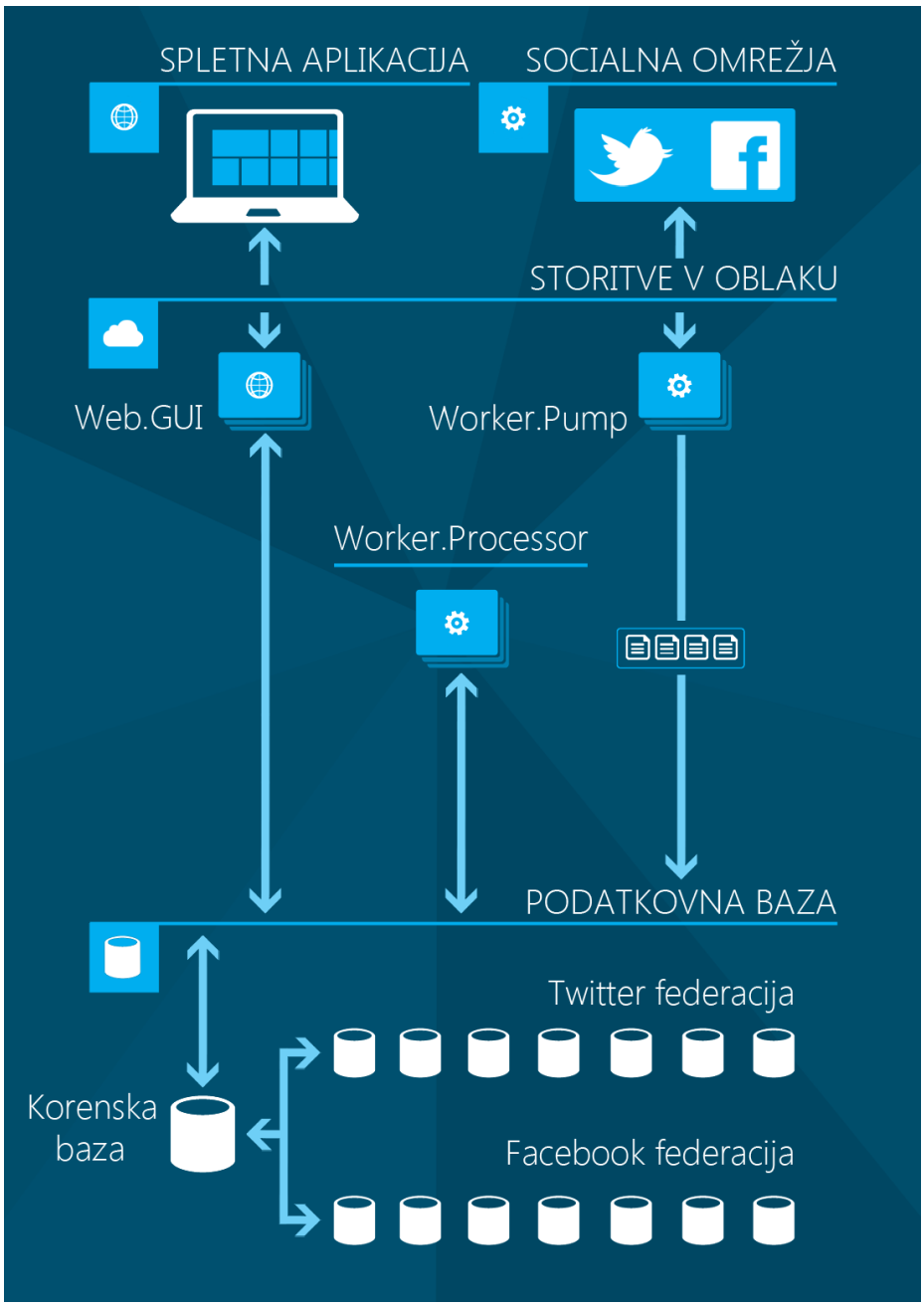
Vse tri vloge so zasnovane tako, da jih je mogoče skalirati tako vertikalno kot horizontalno in s tem prilagoditi pocesorsko moč glede na trenutno porabo.

Spletna vloga (Web.GUI) vsebuje spletno aplikacijo, ki nam zagotavlja naslednje storitve:

- Ustvarjanje in urejanje iskalnih kriterijev, ki jih želimo spremljati.
- Ustvarjanje in urejanje strani Facebook, ki jih želimo spremljati.
- Vpogled v rezultate in analize.
- Izvoz rezultatov in analiz.
- Procesiranje analiz na zahtevo.
- Administracijo nad celotno aplikacijo.
- Vmesnik API za dostop do rezultatov in analiz.

Vloga črpalke (Worker.Pump) skrbi za:

- Pridobivanje podatkov iz Facebooka in Twitterja na podlagi iskalnih kriterijev in strani Facebook, ki so definirani v bazi.
- Sentimentalno analizo rezultatov.
- Zaznavanje jezika rezultatov.



Slika 4.1: Arhitektura aplikacije

Vloga procesorja (Worker.Processor) skrbi za:

- Procesiranje analize zadetkov.
- Procesiranje analize dosega.
- Procesiranje sentimentalne analize.
- Brisanje starih rezultatov in analiz.
- Brisanje iskalnih kriterijev in strani Facebook.

Podatkovna baza je sestavljena iz korenske baze, federacije Twitter in federacije Facebook. V korenski bazi so shranjeni podatki o uporabnikih, sistemskih nastavitvah in sistemski dnevnik. Federacija Twitter hrani iskalne kriterije, rezultate in analize, ki so vezani nanje. Federacija Facebook pa hrani strani Facebook, rezultate in analize, ki so vezani nanje.

4.2 Podatkovna baza

Kot je prikazano na sliki 4.1, je podatkovna baza sestavljena iz korenske baze, federacije Twitter in federacije Facebook. Vsaka izmed federacij ima lahko enega ali več federacijskih članov, ki se jih po potrebi doda ali odvzame, glede na potrebe po prostoru ali hitrosti aplikacije.

Povezava do vsakega izmed federacijskih članov poteka preko korenske baze tako, da se najprej povežemo na korensko bazo, nato pa z uporabo ukaza USE FEDERATION podamo bazi ime federacije in ključ porazdelitvene tabele. S tem je odprta povezava do federacijskega člana in lahko nad njim izvršimo poljuben ukaz SQL.

Spodaj je prikazan del metode ExecuteSimpleCommand, ki najprej odpre povezavo na korensko bazo. Če sta kot parametra metode podana federacija in federacijski ključ, se na podlagi teh dveh vrednosti odpre povezavo do ustreznega federacijskega člana. Na koncu se izvede ukaz SQL. Na takšen način imamo eno samo metodo za izvajanje ukazov na korenski bazi ali na katerem izmed federacijskih članov. Na podoben način so v aplikaciji implementirane vse metode za delo s podatkovno bazo.

```
OpenConnection();
// povezava na federacijo
```

```
string federationString = null;
switch (federation)
{
   case Federation.SearchCriteria:
      federationString = String.Format("USE FEDERATION
          SearchCriteriaFederation(scGuid='{2}') WITH RESET,
          FILTERING=OFF",
      ((Guid)federationKey).ToString());
   break;
   case Federation.FacebookPage:
      federationString = String.Format("USE FEDERATION
          FacebookPageFederation(fpGuid='{2}') WITH RESET,
          FILTERING=OFF",
      ((Guid)federationKey).ToString());
   break;
}
using (SqlCommand fedCmd = new SqlCommand(federationString, Connection))
{
   fedCmd.TryExecuteNonQuery();
}
// izvrsitev ukaza
SqlCommand objCommand = new SqlCommand(commandText, Connection);
objCommand.TryExecuteNonQuery();
CloseConnection();
```

V primeru, ko želimo pridobiti podatke iz vseh federacij, uporabimo razpršene (angl. fan-out) poizvedbe. Spodaj je prikazan del metode, ki skrbi za pridobivanje podatkovne tabele iz celotne federacije glede na podan ukaz. Metoda preveri, če je podana federacija Twitter ali Facebook in se na podlagi tega vzporedno poveže na vsakega izmed federacijskih članov ter iz njega pridobi podatke tako, da nad njim izvede podan ukaz SQL. Podatke nato združi v spremenljivki, ki jo med združitvijo podatkov zaklepa, da ne bi prišlo do napake, ko bi jo niti poskušale hkrati spreminjati. Na koncu metoda vrne skupek podatkov, ki jih je pridobila iz vseh federacijskih članov.

```
DataTable data = new DataTable();
switch (federation)
{
```

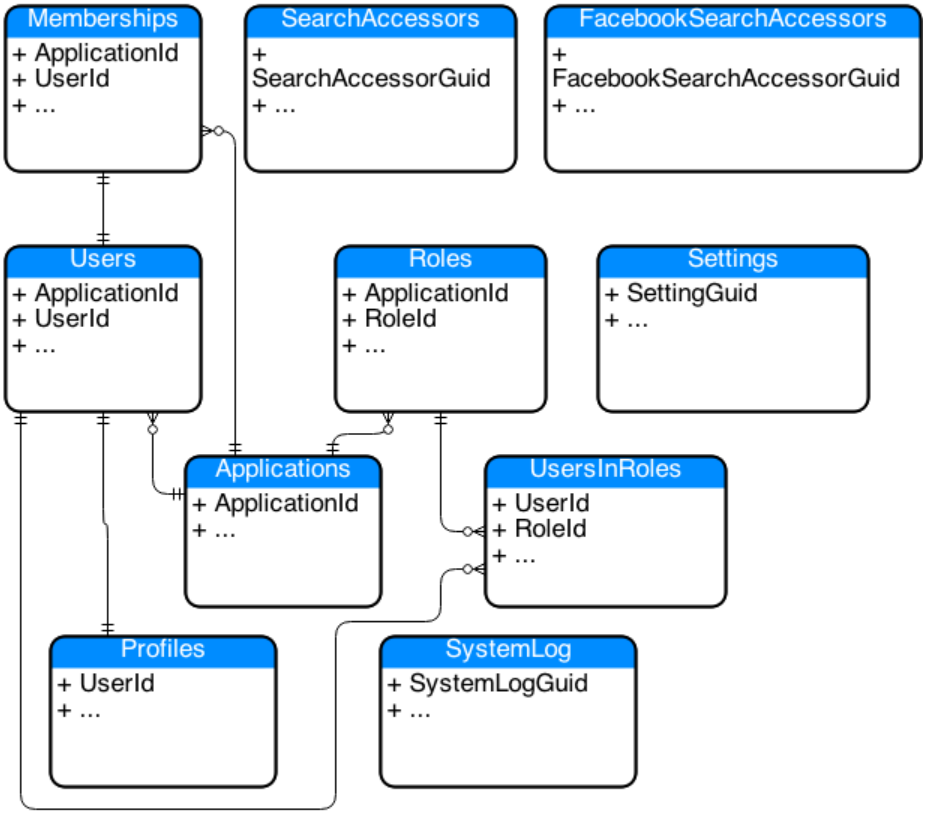
```
   case Federation.SearchCriteria:
      Parallel.ForEach(SearchCriteriaFederation.Ranges,
          delegate(FederationRange<Guid> fed)
      {
         Database database = new Database();
         DataTable table = database.GetDataTable(command, federation,
             fed.RangeLow, false)
         lock (data) data.Merge(table, true, MissingSchemaAction.Add);
      });
      break;
   case Federation.FacebookPage:
      Parallel.ForEach(FacebookPageFederation.Ranges,
          delegate(FederationRange<Guid> fed)
      {
         Database database = new Database();
         DataTable table = database.GetDataTable(command, federation,
             fed.RangeLow, false)
         lock (data) data.Merge(table, true, MissingSchemaAction.Add);
      });
}
return data;
```

4.2.1 Korenska baza

V korenski bazi so shranjeni podatki o uporabnikih aplikacije, nastavitvah aplikacije in sistemski dnevnik.

Tabele o aplikaciji in njenih uporabnikih so samodejno zgenerirane s strani razreda MembershipProvider, ki je del ogrodja .NET. Podatki o aplikaciji so shranjeni v tabeli Applications. Podatki o uporabnikih so shranjeni v tabelah Users, Membership in Profiles. Vloge uporabnikov so definirane v tabeli Roles, povezava med vlogami in uporabniki pa je definirana v tabeli UsersInRoles. Povezave med tabelami so prikazane na sliki 4.2.

Podatki za dostop do družbenih omrežij so shranjeni v tabelah SearchAccessors in FacebookSearchAccessors, nastavitve aplikacije v tabeli Settings, sistemski dnevnik pa v tabeli SystemLog.



Slika 4.2: Diagram korenske podatkovne baze

4.2.2 Federacija Twitter

V federaciji Twitter so shranjene definicije iskalnih kriterijev, rezultati, pridobljeni iz družbenih omrežij, analize zadetkov, analize dosega, analize sentimenta ter nastavitve nad iskalnimi kriteriji.

Definicije iskalnih kriterijev, na podlagi katerih se pridobivajo rezultati in kreirajo analize, so shranjene v tabeli SearchCriteria. Prioritetni urniki iskalnih kriterijev, ki povedo črpalki, kdaj ima kriterij višjo prioriteto, so shranjeni v tabeli SearchCriteriaSchedules.

Slika 4.3: Diagram federacije Twitter

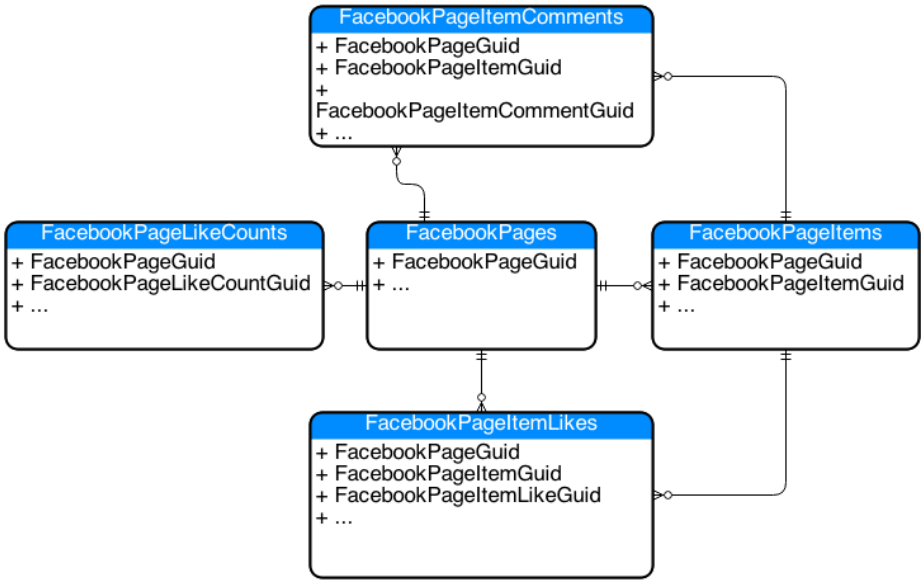
Rezultati, ki jih vloga črpalke pridobi s Twitterja skupaj z njihovo sentimentalno analizo, so shranjeni v tabeli Tweets, omembe uporabnikov v tabeli TweetMentions, oznake v tabeli TweetHashTags, hiperpovezave v tabeli TweetLinks, slike in ostala medijska vsebina pa v tabeli TweetMedia. Ročno označeni rezultati so shranjeni v tabeli ResultsAnnotations.

Analiza zadetkov, ki jo na podlagi rezultatov izračuna vloga procesorja, je shranjena v tabeli SearchCriteriaHitCounts, analiza dosega v tabeli SearchCriteriaReachCounts, analiza sentimenta pa v tabeli SearchCriteriaSentimentCounts.

Kot porazdelitvena tabela federacije je izbrana tabela SearchCriteria, saj so na podlagi le-te podatki najbolj enakomerno porazdeljeni. Ključ porazdelitvene tabele je searchCriteriaGuid, ki je tipa uniqueidentifier (GUID), kar omogoča enostavno genenriranje ključev, ki se enakomerno porazdelijo po federacijah. Zaradi zahtev platforme Windows Azure mora biti porazdelitveni ključ v vseh

tabelah, ki so znotraj federacije, prav tako mora biti nad tem ključem in vsakim posameznim ključem tabele zgeneriran indeks. Povezave med tabelami so prikazane na sliki 4.3.

4.2.3 Federacija Facebook



Slika 4.4: Diagram federacije Facebook

V federaciji Facebook so shranjene definicije strani Facebook, objave uporabnikov in strani, komentarji in všečki na te objave ter analiza všečkov.

Definicije strani Facebook, na podlagi katerih se pridobivajo objave, komentarji in všečki, so shranjene v tabeli FacebookPages.

Objave so shranjene v tabeli FacebookPageItems, komentarji na objave v tabeli FacebookPageItemComments, všečki pa v tabeli FacebookPageItemLikes.

Analiza všečkov strani, ki jo vloga procesorja računa sproti na podlagi pridobljenih podatkov, je shranjena v tabeli FacebookPageLikeCounts.

Kot porazdelitvena tabela je izbrana tabela FacebookPages, kot porazdelitveni ključ tabele pa facebookPageGuid, ki je tako kot v federaciji Twitter prisoten v vseh tabelah in je nad njim in drugim ključem tabele zgeneriran indeks. Povezave med tabelami so prikazane na sliki 4.4.

4.3 Vloga črpalke

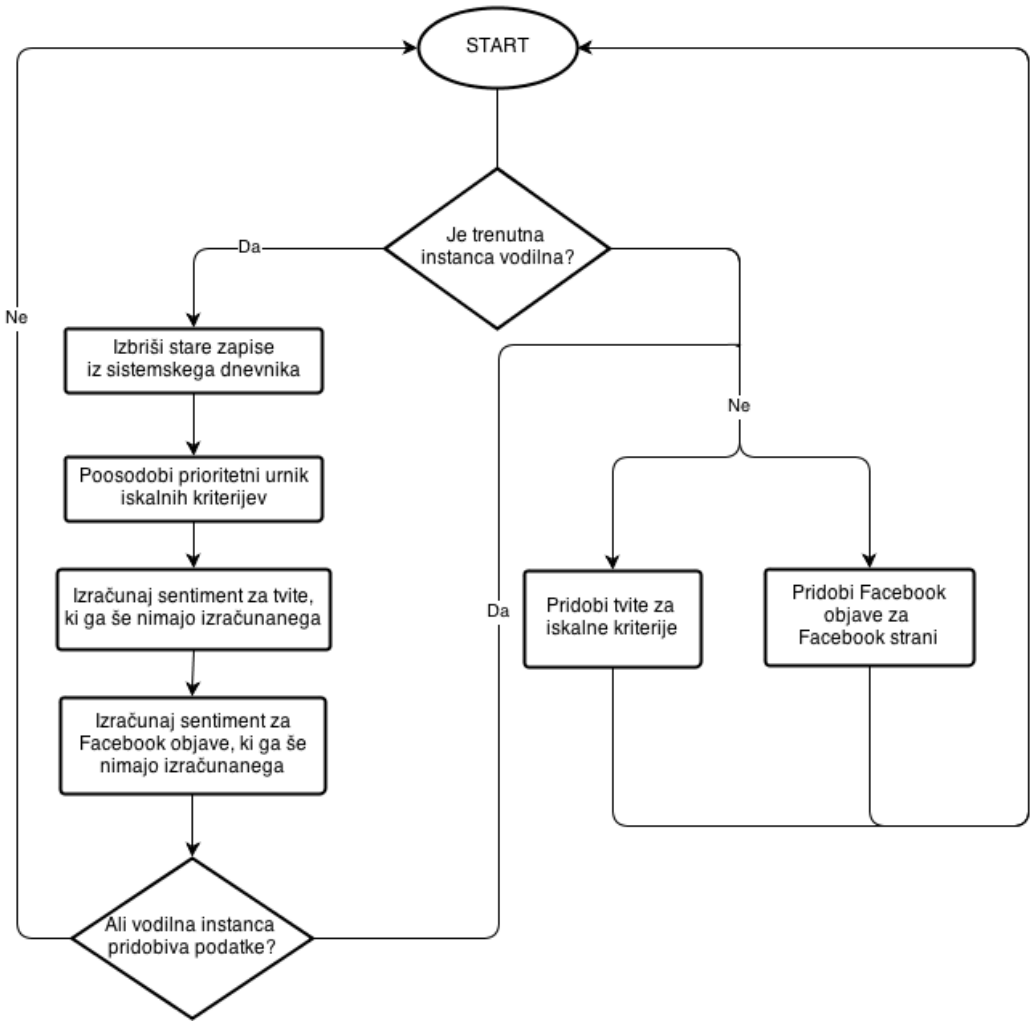
Primarna naloga vloge črpalke je pridobivanje rezultatov iz družbenih omrežij Twitter in Facebook glede na definicije iskalnih kriterijev in strani Facebook ter analiza sentimenta pridobljenih rezultatov. Poleg tega vloga skrbi tudi za brisanje starih zapisov iz sistemskega dnevnika in posodabljanje prioritetnega urnika iskalnih kriterijev.

Operacije znotraj vloge so implementirane idempotentno, kar pomeni, da lahko ista koda teče na večih instancah hkrati in to ne vpliva na končni rezultat. To nam omogoča enostavno horizontalno skaliranje, odpornost na napake ...

Kot je razvidno s slike 4.5, se instanca takoj po zagonu vpraša, če je vodilna instanca. V tem koraku se izvede tudi izbira nove vodilne instance v primeru, da je trenutni vodja neodziven ali če se odziva prepočasi. Če je instanca vodilna, prične z brisanjem sistemskega dnevnika, nato nadaljuje s posodobitvijo prioritetnih urnikov iskalnih kriterijev, asinhronim izračunom sentimenta za rezultate iz omrežja Twitter, ki niso bili izračunani sinhrono, in na koncu še z izračunom sentimenta za rezultate iz omrežja Facebook, ki niso bili izračunani sinhrono.

Če instanca ni vodilna ali če je v nastavitvah definirano, da vodilna instanca opravlja tudi delo črpalke, prične z vzporednim pridobivanjem podatkov tako iz omrežija Twitter kot tudi iz omrežja Facebook v ločenih nitih. Po končanem delu se instanca vrne na začetek in tako teče in pridobiva podatke v neskončni zanki.

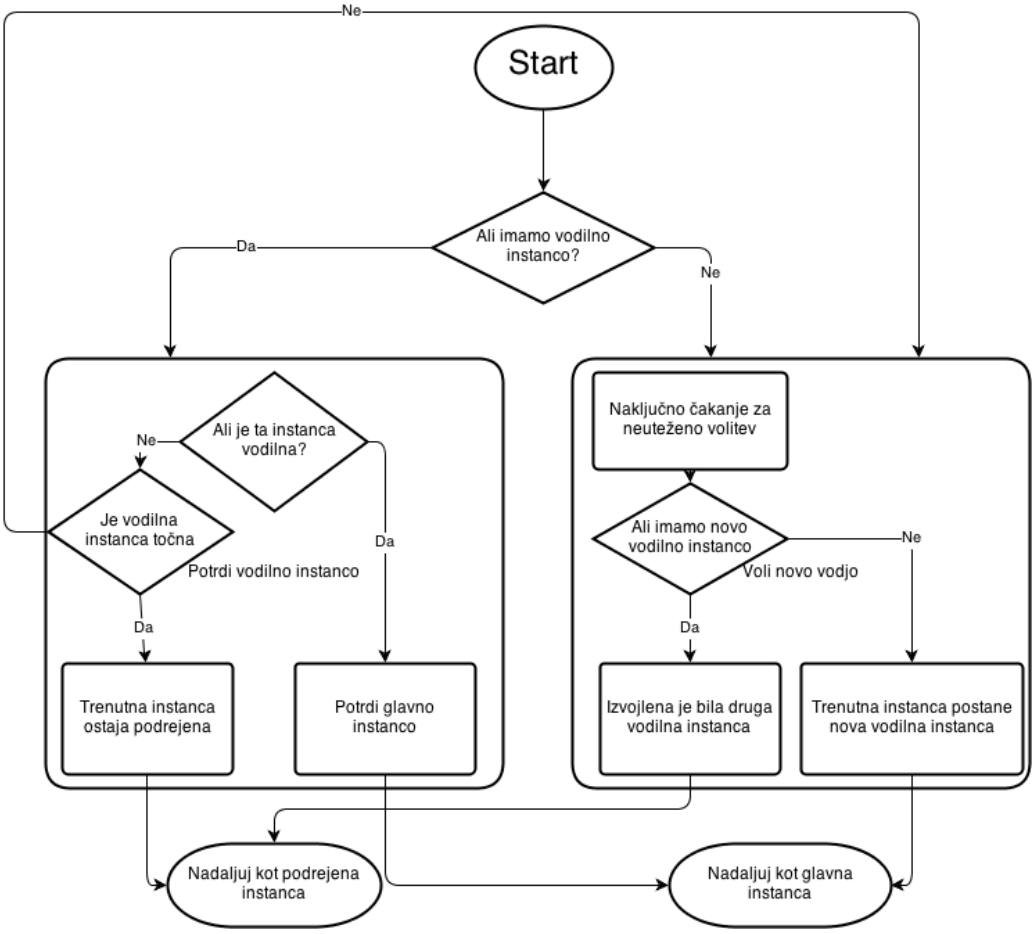
Analiza sentimenta se nad rezultati izvaja sinhrono takoj po njihovi pridobitvi. Asinhrona analiza sentimenta, ki jo opravlja vodila instanca, pride v poštev le, če je prišlo do napake pri izvajanju katere izmed instanc v vlogi črpalke.



Slika 4.5: Diagram poteka vloge črpalke

4.3.1 Izbira in delo vodilne instance

Pri izbiri vodilne instance uporabljamo spremenjeno verzijo algoritma Bully. Glavna razlika med algoritmoma je ta, da se pri naši implementaciji algoritma ne uporablja prioritetnih številk instanc, temveč se vodjo izbere po principu "prvi pride prvi melje".



Slika 4.6: Diagram algoritma za izbiro vodilne instance

Algoritem poteka tako, da se instanca najprej vpraša, če je že izbrana vodilna instanca. Če vodilna instanca obstaja, se sproži proces potrditve instance, v nasprotnem primeru se sproži proces izbora nove vodilne instance.

Potrditev vodilne instance poteka tako, da se instanca najprej vpraša, če je sama vodja, in v tem primeru potrdi vodilno instanco ter nadaljuje z delom kot vodilna instanca. Če je vodilna instanca katera izmed preostalih instanc, trenutna instanca preveri, če je le-ta točna. Če je, trenutna instanca nadaljuje z delom kot podrejena, v nasprotnem primeru sproži proces izbora nove instance.

Proces izbora nove vodilne instance poteka tako, da na začetku vsaka in-

stanca čaka naključno število milisekund z namenom preprečitve istočasnega imenovanja vodilne instance. Po čakanju se vsaka instanca vpraša, če že obstaja vodilna instanca. Če vodilna instanca ne obstaja, prevzame trenutna instanca delo vodilne, sicer nadaljuje delo kot podrejena instanca. Diagram poteka algoritma je prikazan na sliki 4.6.

Vodilna instanca po končanem izboru prične s svojim delom. Najprej počisti stare zapise iz sistemskega dnevnika tako, da na korenski bazi požene proceduro, ki izbriše stare zapise.

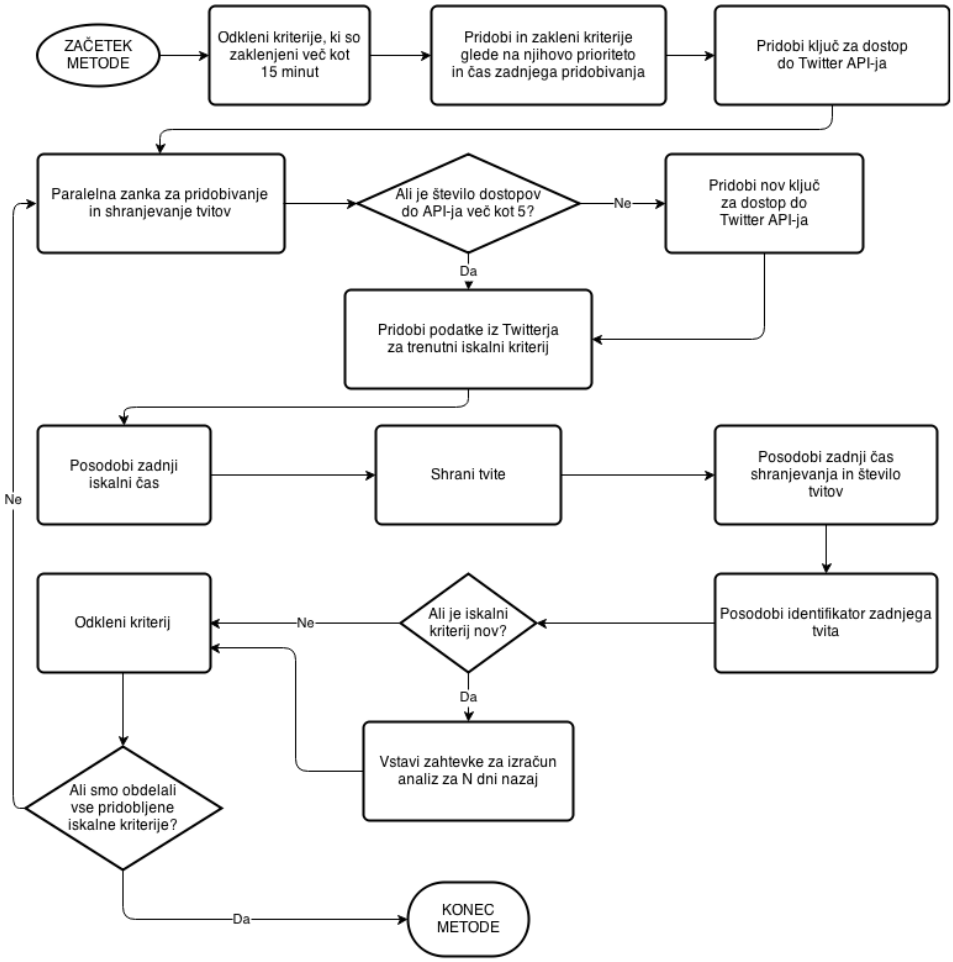
Nato nastavi prioritete iskalnih kriterijev tako, kot je zapisano v prioritetnem urniku. To stori tako, da vsako minuto požene metodo za posodabljanje prioritetnih urnikov na vsakem članu federacije Twitter. Metoda se poveže na federacijskega člana in iz njega prebere zapise prioritetnih urnikov, ki so aktivni v danem trenutku in še niso bili aktivirani, ter nastavi prioritete iskalnih kriterijev na vhodno vrednost. Prioritetni urnik nato označi kot aktiviran. V naslednji stopnji nastavi prioriteto iskalnih kriterijev na izhodno vrednost za kriterije, ki so jim prioritetni urniki potekli, ter izbriše prioritetne urnike.

Ko vodilna instanca zaključi s posodabljanjem prioritetnih urnikov, prične z asinhronim računanjem sentimenta za tvite, ki še nimajo izračunanega sentimenta. Nalogo izvaja periodično tako, da se poveže na vsakega izmed federacijskih članov in iz njega pridobi tvite, ki še nimajo izračunanega sentimenta. Tvite nato pretvori v seznam serializabilnih objektov, s katerim kliče storitev za izračun sentimenta, ki jo je razvil Institut Jožefa Stefan in je opisana v poglavju 4.3.4. Po tem, ko storitev izračuna sentiment za podane tvite, instanca shrani rezultate v podatkovno bazo.

Naslednja naloga vodilne instance je, da asinhrono izračuna sentiment za objave in komentarje Facebook, ki še nimajo izračunanega sentimenta. Nalogo izvaja periodično tako, da se poveže na federacijskega člana, iz njega pridobi objave Facebook, ki še nimajo izračunanega sentimenta, jih pretvori v seznam serializabilnih elementov ter z njim kliče storitev za izračun sentimenta. Ko storitev izračuna sentiment za podane elemente, metoda shrani rezulate v podatkovno bazo. V drugem delu metoda isto stvar ponovi še za komentarje Facebook.

4.3.2 Pridobivanje in shranjevanje podatkov iz omrežja Twitter

Za pridobivanje in shranjevanje podatkov iz družbenega omrežja Twitter skrbi metoda Harvest. Slika 4.7 prikazuje diagram poteka, ki ga implementira metoda Harvest.



Slika 4.7: Diagram algoritma za pridobivanje in shranjevanje podatkov iz omrežja Twitter.

V družbenem omrežju Twitter so vsi podatki shranjeni v entiteti tvit, kar

omogoča hitro in enostavno pridobivanje in shranjevanje pridobljenih podatkov. Za pridobivanje podatkov uporabljamo vmesnik Twitter API, ki nam vrača podatke v obliki JSON, ki jih pretvorimo v objekte .NET s pomočjo knjižnice Newtonsoft.JSON.

Metoda Harvest na začetku odklene vse iskalne kriterije, ki so bili zaklenjeni več kot petnajst minut, in tako prepreči, da bi kateri izmed iskalnih kriterijev ostal večno zaklenjen, če bi prišlo do napake pri izvajanju metode ali bi se instanca ustavila sredi dela zaradi zunanjih vzrokov.

Nato pridobimo določeno število iskalnih kriterijev ter jih zaklenemo, da jih ne bi hkrati procesirala katera izmed preostalih instanc vloge. To storimo tako, da iskalne kriterije istočasno zaklenemo in pridobimo iz baze. Iskalne kriterije se izbere glede na njihov zadnji čas iskanja in prioriteto.

Če trenutno ni nobenega odklenjenega kriterija za obdelavo, metoda konča z delom in miruje deset sekund.

V primeru, da smo uspešno pridobili iskalne kriterije, pridobimo iskalni ključ za dostop do vmesnika Twitter API. Ključ pridobimo z metodo, ki najprej ponastavi število preostalih klicev nad ključi, ki jim je že pretekel čas ponastavitve, ter nato izbere iskalni ključ z največjim številom preostalih klicev do vmesnika API. Trenutna omejitev je stoosemdeset klicev v petnajstih minutah.

Po uspešno pridobljenem ključu nadaljujemo z delom nad iskalnimi kriteriji tako, da hkrati paralelno obdelujemo večje število iskalnih kriterijev. Stopnja paralelnosti je definirana v nastavitvah. Takoj na začetku preverimo število preostalih klicev do vmesnika Twitter API ter po potrebi zamenjamo ključ. Če se število preostalih klicev na vseh ključih drastično zmanjša, povečamo čas mirovanja instanc med iskanji, kar omogoči, da se klici na ključih ponastavijo.

V naslednjem koraku z uporabo metode Search v vmesniku Twitter API pridobimo tvite za iskalni kriterij, tako da v zanki kličemo Twitter Search API z iskalnim pogojem ter identifikacijsko številko zadnjega pridobljenega tvita. API nam vrne največ sto rezultatov hkrati. Zanko prekinemo, ko nam API ne vrne več nobenega tvita oziroma prekoračimo največjo dovoljeno število ponovitev, ki je definirana v nastavitvah. Na koncu metode kličemo še servis za izračun sentimenta in jezika s pridobljenimi tviti.

Po uspešno zaključeni metodi za pridobitev tvitov posodobimo zadnji iskalni čas iskalnega kriterija.

Nato pride na vrsto filtriranje tvitov na podlagi filtrov, ki so definirani za

vsak iskalni kriterij. Obstajajo filter uporabnikov, filter jezikov, filter besed in filter črk. Filtri uporabnikov, besed in črk odstranijo tvite, ki zadostujejo pogojem, ki so definirani v filtrih, filter jezikov pa obdrži le tvite, ki so napisani v enem izmed jezikov, definiranih v filtru.

Filtriranju sledi shranjevanje, ki se izvaja z ukazom BULK INSERT, ki vstavi celotno tabelo zapisov v podatkovno bazo, kar je veliko hitrejše kot shranjevanje vsakega zapisa posebej. Shranjevanje poteka tako, da iz baze najprej pridobimo definicije za tabele, nato vanje vstavimo tvite ter podatke, ki sodijo zraven. Vsakih N tvitov, kjer je N nastavljiv, shranimo podatke v bazo in počistimo tabele. Ko smo prišli čez vse pridobljene tvite v trenutni iteraciji, shranimo še preostanek tabele v bazo. Po vsakem shranjevanju tvitov v bazo posodobimo zadnji čas shranjevanja, število tvitov in identifikator zadnjega shranjenega tvita za iskalni kriterij.

V primeru, da je iskalni kriterij nov, vstavimo zahtevke za analize za preteklo obdobje, za katerega že imamo rezultate. Zahtevki za analize in njihov izračun so bolj podrobno opisani v poglavju 4.4.

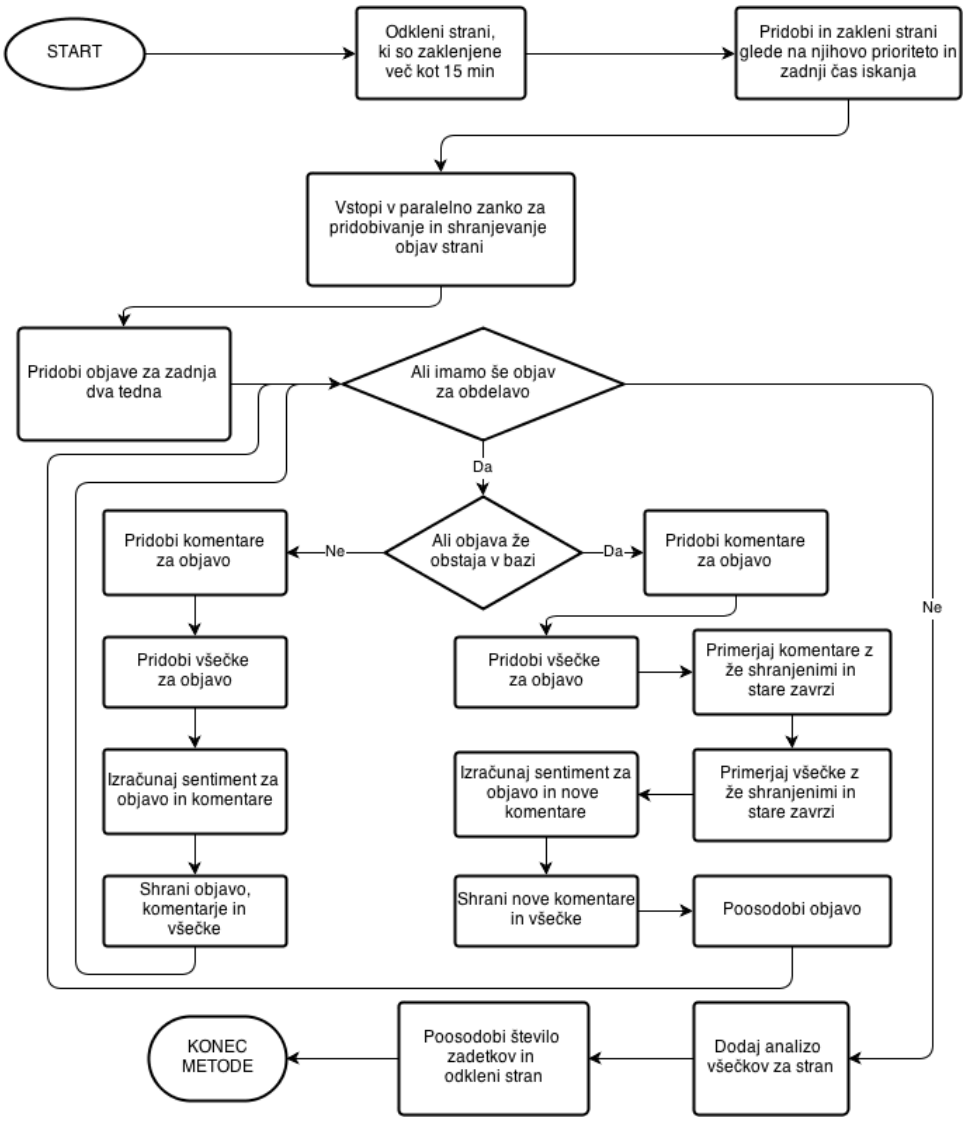
Na koncu paralelne zanke še odklenemo iskalni kriterij in zaključimo delo.

4.3.3 Pridobivanje in shranjevanje podatkov iz omrežja Facebook

Za pridobivanje in shranjevanje podatkov iz družbenega omrežja Facebook skrbi metoda Harvest. Okvirni diagram poteka metode je prikazan na sliki 4.8.

Metoda je v osnovi dokaj podobna tisti, ki skrbi za pridobivanje podatkov iz družbenega omrežja Twitter, vendar je zaradi bolj zapletene strukture objav, komentarjev in všečkov na Facebooku pridobivanje in shranjevanje podatkov drugačno. Vsaka objava ima všečke in komentarje, kar pomeni, da ni dovolj le pridobiti objavo s Facebooka in jo shraniti, temveč moramo ves čas spraševati Facebook, če ima objava kakšen nov komentar ali všeček, in to zabeležiti v podatkovni bazi. Takšen način pridobivanja podatkov je veliko bolj zamuden, kar se tudi pozna na času, potrebnem za obdelavo strani Facebook.

Za pridobivanje podatkov s Facebooka uprabljamo vmesnik Facebook Graph API, do katerega dostopamo s ključem, ki ga pridobimo pri registraciji aplikacije. Facebook Graph API nam tako kot Twitter API podatke vrača v obliki JSON. Zaradi Facebookove varnostne politike nam za naše potrebe pridobivanja podatkov zadostuje zgolj en ključ za dostop do vmesnika API.



Slika 4.8: Diagram algoritma za pridobivanje in shranjevanje podatkov iz omrežja Facebook.

Metoda Harvest na začetku odklene strani Facebook, ki so bile zaklenjene za več kot petnajst minut, in tako prepreči, da bi katera izmed strani ostala zaklenjena v primeru napake. Nato sledi pridobivanje in zaklepanje strani Facebook za obdelavo, kar poteka na podoben način kot pri pridobivanju podatkov

iz družbenega omrežja Twitter.

Po uspešno zaključenem pridobivanju strani pričnemo z njihovo obdelavo v paralelni zanki, katere stopnja paralelnosti je določena v nastavitvah. V zanki obdelamo vsako stran posamezno, tako da najprej pridobimo strani iz omrežja Facebook za zadnja dva tedna ter nato za vsako posamezno objavo preverimo, če jo imamo že shranjeno v bazi.

V primeru, da objave še nismo shranili v podatkovni bazi, zanjo pridobimo vse komentarje in všečke. Nato objavo in njene komentarje pošljemo servisu za izračun sentimenta. Po pridobljenem sentimentu pričnemo s shranjevanjem objave. To storimo tako, da najprej z ukazom INSERT shranimo objavo, nato pa z uporabo ukaza BATCH INSERT shranimo še vse komentarje in všečke, po tisoč naenkrat.

V primeru, da smo objavo že shranili v podatkovno bazo, poteka shranjevanje objave na nekoliko drugačen način. Prav tako najprej pridobimo komentarje in všečke s Facebooka, nato pa posodobimo objavo v podatkovni bazi z ukazom UPDATE. Komentarje za objavo shranimo tako, da najprej pridobimo že shranjene komentarje iz podatkovne baze in jih primerjamo s komentarji, pridobljenimi s Facebooka. Komentarje, ki še niso shranjeni v podatkovni bazi, shranimo z ukazom BATCH INSERT po tisoč naenkrat. Isti postopek ponovimo še za všečke.

Ko na tak način obdelamo vse objave, pridobljene s Facebooka, dodamo v bazo zapis, ki bo služil za analizo všečkov strani. Analiza všečkov strani nam pove, kako so se všečki strani spreminjali skozi čas.

Na koncu paralelne zanke posodobimo zadnji čas iskanja na strani in stran odklenemo.

4.3.4 Detekcija jezika in analiza sentimenta

Za detekcijo jezika in analizo sentimenta skrbi servis, ki ga je razvil Institut Jožef Stefan. Servis trenutno podpira naslednje jezike: slovenski, angleški, hrvaški, srbski, bosanski, albanski in bolgarski. V izdelavi so tudi modeli za ruski, kazahstanski in portugalski jezik.

Detekcija jezika in analiza sentimenta deluje na podlagi strojnega učenja, tako da na podlagi ročno označenih rezultatov izdela jezikovni model, ki vsebuje pravila, s pomočjo katerih nato določi jezik in sentiment. Pravila so definirana na pozitivnosti in negativnosti besed in besednih zvez. Bolj kot je neka beseda

pozitivna ali negativna, večjo težo ima.

Za začetni model potrebujemo najmanj petdeset tisoč razultatov z ročno označenim sentimentom. Delovanje modela izboljšamo tako, da rezulate, pri katerih je model napačno zaznal jezik in sentiment, ročno popravimo.

Sentiment s servisa pridobimo tako, da kličemo metodo z rezultati, ki jih pridobimo s Facebooka ali Twitterja. Metoda nam kot rezultat vrne seznam objektov, v katerem so shranjeni podatki o zaznanem jeziku in sentimentu. Poleg teh podatkov dobimo še podatke o zanesljivosti detekcije ter niz, ki nam pove, katere besede so vplivale na analizo sentimenta. Če je zanesljivost sentimenta pod določeno mejo, obravnavamo rezultat kot nevtralen. Spodaj je prikazana metoda, ki kliče servis za detekcijo jezika in analizo sentimenta.

```
public static List<ItemSentimentResult>
    GatherSentiment(List<LightweightItem> items){
  if (proxy == null) {
    proxy = new TweetSentimentService.TweetSentimentService();
  }
  List<ItemSentimentResult> sentiments = proxy.GetItemSentiments(items);
  return sentiments;
}
```

4.4 Vloga procesorja

Primarna naloga vloge procesorja je kreiranje zahtevkov za analize nad iskalnimi kriteriji in njihov izračun. Poleg tega skrbi vloga še za brisanje starih analiz in rezultatov ter za uničenje iskalnih kriterijev in strani Facebook, ki jih je uporabnik izbrisal preko uporabniškega vmesnika.

Prav tako kot operacije znotraj vloge črpalke so tudi operacije znotraj vloge procesorja implementirane idempotentno.

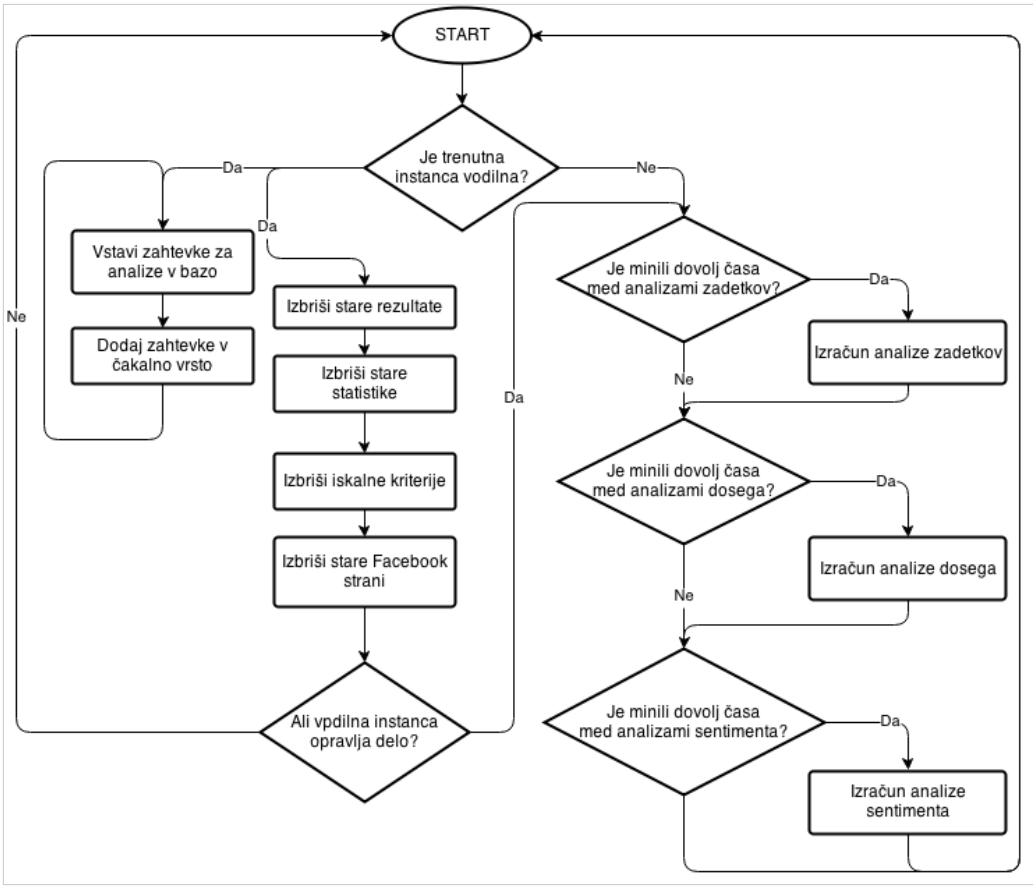
Kot je razvidno s slike 4.9, delo vloge poteka tako, da se instanca najprej vpraša, če je vodilna. V tem koraku se prav tako izvede izbor nove vodilne instance, če je vodilna instanca neodzivna oziroma se odziva prepočasi.

Vodilna instanca prične s svojim delom tako, da v ločeni niti vstavi zahtevke za analize v bazo in zatem še v čakalno vrsto za procesiranje.

V glavni niti medtem vodilna instanca na vsake N ur, kjer je N nastavljiv,

izbriše stare rezultate in analize iz federacije Twitter in Facebook. Zatem uniči še iskalne kriterije in strani Facebook, ki jih je uporabnik izbrisal iz spletne aplikacije.

Če instanca ni vodilna oziroma če je v nastavitvah definirano, da vodilna instanca opravlja tudi procesiranje analiz, le-ta prične procesirati analize iz čakalne vrste, v katero jih je vstavila vodilna instanca. Ker se analize procesirajo v določenih intervalih, se pred vsakim procesiranjem vprašamo, če je minilo dovolj časa. Na takšen način procesiramo analize zadetkov, dosega in sentimenta za iskalne kriterije. Analize za strani Facebook se izračunajo na zahtevo uporabnika in zanje ni odgovorna ta instanca.



Slika 4.9: Diagram poteka vloge procesorja

4.4.1 Izbira in delo vodilne instance

Za izbiro vodilne instance se uporablja isto metodo kot za izbiro vodilne instance v vlogi črpalke, ki je opisana v poglavju 4.3.1.

Vodilna instanca nato zažene proces v novi niti, ki v neskončni zanki skrbi za generiranje zahtevkov za analize iskalnih kriterijev, da ne bi zaradi kakšne zamudne operacije, kot je brisanje, prišlo do daljšega obdobja, v katerem ne bi generirali zahtevkov in tako ne bi imeli konsistentnih analiz.

Analize se kreirajo za različna časovna okna, od ene minute do enega tedna. Velikost časovnega okna nam pove, za koliko časa nazaj upoštevamo rezultate za zapis v tabeli analiz. Glede na velikost časovnega okna in prioriteto iskalnega kriterija se razlikujejo tudi časovne periode med zapisi v tabelah analiz. Za časovna okna med eno minuto in eno uro se za iskalne kriterije z normalno prioriteto analize kreirajo na petnajst minut, pri čemer se za visokofrekvenčne kriterije analize med eno minuto in dvemi urami kreirajo na minuto. Periode med analizami se povečujejo z velikostjo časovnega okna. Za sedemdnevno analizo se za iskalni kriterij z normalno prioriteto analize kreirajo na vsakih šest ur. Periode je mogoče spreminjati v tabeli nastavitev, časovna okna analiz so fiksna.

Analiza zadetkov nam pove, koliko rezultatov je bilo v času meritve za obdobje časovnega okna. To pomeni, da če bi meritev potekala sedaj in bi bilo časovno okno meritve ena ura, bi aplikacija preštela vse rezultate za zadnjo uro ter rezultat shranila v bazo. Analiza dosega vsebuje absolutni doseg rezultatov, kar nam pove, koliko uporabnikov so dosegli rezultati, število uporabnikov, ki so te rezultate kreirali, in število krajev, iz katerih so bili rezultati poslani. Analiza sentimenta meri število pozitivnih, negativnih in nevtralnih sentimentov.

Generiraje zahtevkov za analize poteka tako, da se v zanki sprehodimo čez vse federacijske člane federacije Twitter in za vsako prioriteto iskalnih kriterijev poženemo metodo, ki generira zahtevke, jih shrani v bazo in doda v čakalno vrsto. Aplikacija pozna pet tipov prioritet iskalnih kriterijev: najnižja prioriteta, nizka prioriteta, normalna prioriteta, visoka prioritera in najvišja prioriteta.

Metoda najprej preveri, če je minilo zadosti časa med analizami za podano prioriteto, nato kliče metodo za generiranje zadetkov za analizo zadetkov, analizo dosega in analizo sentimenta.

Nato iz podane federacije pridobi vse kriterije s podano prioriteto in za vsak kriterij generira zahtevke, ki jih hrani v začasni spremenljivki. Ko konča z

generiranjem zahtevkov, tabelo shrani v podatkovno bazo z ukazom BATCH INSERT, zatem pa pošlje identifikatorje zahtevkov v čakalno vrsto, kjer čakajo na procesiranje ene izmed instanc. Pri generiranju zahtevka v tabelo shranimo identifikator zahtevka, čas analize, velikost časovnega okna in identifikator iskalnega kriterija.

Vzporedno z generiranjem zahtevkov za analize mora vodilna instanca poskrbeti tudi za brisanje rezultatov, analiz, iskalnih kriterijev in strani Facebook. To poteka v glavni niti programa v neskončni zanki.

V zanki gre instanca najprej čez vse federacijske člane federacije Twitter. Za vsakega izmed federacijskih članov preveri, če je že minilo dovolj časa od zadnjega brisanja rezultatov in statistik, in požene metodo, ki na podatkovni bazi izvede procedure za brisanje rezultatov in analiz, starejših od podanega števila dni. Brisanje rezultatov in analiz vodilna instanca izvaja na vsakih šest ur.

Naslednja naloga vodilne instance na federaciji Twitter je, da izbriše iskalne kriterije, ki jih je uporabnik v aplikacij označil kot izbrisane, in vse zapise, vezane nanje. Nalogo brisanja se prepusti tej vlogi zato, ker je brisanje zaradi prevelikega števila rezultatov prepočasno, da bi ga opravljala spletna vloga na zahtevo uporabnika. Brisanje iskalnih kriterijev poteka tako, da metoda najprej izbere vse iskalne kriterije iz trenutnega federacijskega člana in za vsakega kliče proceduro SQL, ki nato izvede brisanje.

Po končanem brisanju rezultatov, analiz in iskalnih kriterijev iz federacije Twitter prične vodilna instanca brisati razultate, analize in strani Facebook iz federacije Facebook. Podobno kot prej se instanca sprehodi čez vsakega izmed federacijskih članov in nad njim izvede brisanje.

Na vsakih šest ur se instanca sprehodi čez vse strani Facebook na trenutnem federacijskem članu in za vsako stran izvede proceduro za brisanje starih rezultatov in analiz.

Enkrat na dan instanca izbriše strani Facebook, ki jih je uporabnik označil za brisanje. To stori tako, da iz federacijskega člana pridobi vse strani Facebook, označene za brisanje, in za vsako izvede proceduro SQL, ki poskrbi, da se stran in nanjo vezani zapisi izbrišejo iz baze.

Če vodilna insanca ni določena za izračun analiz, je s tem zaključila svoj cikel dela in po določenem času mirovanja cikel prične znova. Če pa je vodilna insanca določena, da sama opravlja tudi računanje analiz, prične s procesiranjem

analiz, kot je opisano v naslednjem poglavju.

4.4.2 Procesiranje zahtevkov za analize

Procesiranje zahtevkov za analize lahko opravljajo vse instance, če je tako določeno v nastavitvah, v nasprotnem primeru vodilna instanca ne procesira zahtevkov in opravlja le delo vodilne instance.

Vsaka insanca procesira tri različne vrste zahtevkov za analize, in sicer zahtevke za analizo zadetkov, dosega in sentimenta. Med procesiranjem zahtevkov istega tipa mora miniti najmanj pet sekund. Ta čas je mogoče spremeniti v nastavitvah.

Najprej poteka procesiranje zahtevkov za analizo zadetkov, nato analiza dosega in na koncu še analiza sentimenta.

Delovanje metode ProcessCountRequests za zahtevke analize zadetkov je prikazano v spodnjem bloku kode. Ta prikazuje delo s čakalno vrsto, ki jo na vlogi procesorja uporabimo kot mehanizem za upravljanje bremena. Metoda vzame zahtevke za analize iz vrste, v katero jih vstavi vodilna instanca. To stori tako, da v zanki, ki ima število korakov podano kot parameter metode, najprej definira stopnjo paralelnosti, ki določi, v koliko različnih nitih se bo procesiralo zahtevke. Nato se vpraša po tipu analize, ki je podana kot parameter metode. Glede na tip zahtevka pridobi sporočilo z zahtevki za analize iz prave čakalne vrste. Nato prične s procesiranjem zahtevkov v paralelni zanki, kar pohitri instance. Ko sprocesira vse zahtevke, izbriše sporočilo in ves postopek ponovlja, dokler ne izstopi iz glavne zanke. Na koncu vrne število sprocesiranih sporočil. Metoda Count, ki jo ta metoda kliče za procesiranje zahtevka, kliče proceduro SQL, ki prešteje rezultate za iskalni kriterij v časovnem obdobju, definiranem v zahtevku, in seštevek zapiše v bazo.

```
public static int ProcessCountRequests(ProcessingType processingType,
    int maxToProcess){
  int totalCount = 0;
  for (int i = 0; i < maxToProcess; i++){
      CloudQueueMessage queueMessage = null;
      string queueUsed = null;
      string statusString = null;
      int perMessageCount = 0;
      // definiraj stopnjo paralelnosti
```

```
    ParallelOptions parallelOptions = new ParallelOptions();
    parallelOptions.MaxDegreeOfParallelism =
        Settings.DegreeOfProcessingParallelism;
    switch (processingType){
      case ProcessingType.HitCount:
        // pridobi sporocilo iz cakalne vrste
        queueMessage =
            PopMessageByQueuePriority(ProcessingType.HitCount, new
            TimeSpan(1, 0, 0), ref queueUsed);
        if (queueMessage == null) return totalCount;
        // deseriliziraj sporocilo
        ProcessingHitCountRequestMessage hitMessage =
            ProcessingRequestMessage
            .FromMessage<ProcessingHitCountRequestMessage>(queueMessage);
        // sprocesiraj sporocilo
        Parallel.ForEach(hitMessage.SearchCriteriaHitCountGuids,
            parallelOptions, requestGuid => {
          Count(processingType, hitMessage.SearchCriteriaGuid,
              requestGuid);
        });
        perMessageCount += hitMessage.SearchCriteriaHitCountGuids.Count;
        // izbrisi sporocilo
        QueueHelper.DeleteMessage(queueUsed, queueMessage);
        break;
      case ProcessingType.ReachCount:
        ...
      case ProcessingType.SentimentCount:
        ...
    }
    totalCount += perMessageCount;
  }
 }
 return totalCount;
}
```

4.5 Spletna vloga

Windows Azure omogoča spletnim vlogam poganjanje spletnih aplikacij in storitev, samodejno porazdelitev zahtevkov po instancah ter ostale prednosti storitev Azure.

Na vlogi teče spletna aplikacija ASP.NET MVC, ki služi kot spletni vmesnik za uporabnike in kot programski vmesnik (API) za integracijo v druge sisteme IT. Prijava poteka s pomočjo ponudnika ASP.NET Membership, ki na korenski podatkovni bazi avtomatsko zgenerira skripte, potrebne za shranjevanje uporabniških podatkov. Stopnje dostopa uporabnikov se določa z uporabo vlog, kot so administrator, analitik, raziskovalec, osnovni uporabnik in napredni uporabnik.

Zahtevki za pridobivanje strani in podatkov s strežnika potekajo preko nadzornikov. Ti vsebujejo metode, do katerih lahko dostopamo s klici HTTP GET in HTTP POST, parametre metode pa podamo bodisi kot parametre URL pri parametre URL HTTP GET bodisi kot parametre klica HTTP POST. Metoda lahko vrača rezultate različnih tipov. Za strani HTTP uporabimo tip ActionResult, za objekte JSON uporabimo tip JsonResult ... V spodnjem bloku kode sta prikazana dva takšna klica, prvi vrača stran HTTP za prikaz analize zadetkov, drugi vrača objekt JSON s podatki za analizo zadetkov. Atribut Authorize pred metodama dovoli dostop do metode le registriranim uporabnikom.

```
[Authorize]
public ActionResult HitCount()
{
  ViewBag.searchCriteriaGuid = SelectedSearchCriteria.SearchCriteriaGuid;
  return View(PerceptionUser);
}

[Authorize]
public JsonResult GetHitCountJSon(Guid searchCriteriaGuid, String from,
    String to, int hoursBack, int minutesBack)
{
  // Get HitCounts
  SearchCriteria criteria =
      PerceptionUser.GetSearchCriterion(searchCriteriaGuid);
  List<SearchCriteriaHitCounts> hitCounts = criteria.GetHitCounts(from,
```

```
        to, hoursBack, minutesBack)
    HitCountAnalysisJSonWrapper model = new
        HitCountAnalysisJSonWrapper(hitCounts);
    return Json(model, JsonRequestBehavior.AllowGet);
}
```

Za izdelavo strani HTML se uporablja ogrodje Bootstrap, ki ga je razvilo podjetje Twitter in ga kasneje odprlo za javnost. Ogrodje vsebuje kontrole, napisane v jezikih HTML, CSS in JavaScript, med katerimi so gumbi, tabele, izvlečni menuji, galerije ...

Za vizualizacijo podatkov aplikacija uporablja knjižnico D3.js, ki nas ne omejuje na že implementirane funkcionalnosti knjižnice, temveč nam z orodji, ki so napisana z namenom povezave podatkov s prikaznimi elementi, omogoča implementacijo interaktivnih grafov, prilagojenih našim željam.

V spodnjem bloku kode je prikazan primer funkcije Draw, ki s pomočjo knjižnice D3.js nariše črtni grafikon. Metoda najprej v element HTML, ki je shranjen v spremenljivki Container, doda element SVG, nato v element SVG doda risalno površino in jo shrani v spremenljivko Graph. Nato iz objekta JSON, shranjenega v spremenljivki Data, pridobi začetni in končni čas, ki se bo prikazoval na osi x grafa. Nato zgenerira funkcijo X v domeni začetnega in končnega časa v razponu širine elementa HTML, ki hrani graf. Funkcijo Y zgenerira v domeni minimalne in maksimalne vrednosti spremenljivke Count v objektu Data v razponu višine elementa HTTP, ki hrani graf. Zatem ustvari funkcijo Line, ki služi za risanje črte grafa, ta za računanje koordinate x uporabi prej ustvarjeno funkcijo X in za računanje koordinate y funkcijo Y. Na koncu funkcija na risalno površino Graph nariše črto z uporabo funkcije Line, ki ji kot parameter poda podatke v objektu JSON, shranjene v spremenljivki Data.

```
TimeSelectorChart.prototype.Draw = function () {
  // dodaj SVG element na stran
  this.SVG = d3.select("#" + this.Container).append("svg:svg")
      .attr("width", this.GetWidth())
      .attr("height", this.GetHeight());
  // dodaj povrsino za risanje
  this.Graph = this.SVG
      .append("svg:g")
      .attr("transform", "translate(" + this.Margin.Left + "," +
```

```
            this.Margin.Top + ")");
    // pridobi minimalen in maksimalen datum iz objekta JSON
    var startTime = this.Data[0].CreatedDateTimeTimestamp;
    var endTime = this.Data[this.Data.length -
        1].CreatedDateTimeTimestamp;
    // ustvari x in y funkcije za racunanje tock na grafu
    this.X = d3.time.scale().domain([startTime, endTime]).range([0,
        this.Width]);
    this.X.tickFormat(d3.time.format("%Y-%m-%d"));
    this.Y = d3.scale.linear().domain([d3.min(this.Data, function (d) {
        return d.Count; }), d3.max(this.Data, function (d) { return
        d.Count; })]).range([this.Height, 0]);
    // ustvati funkcijo za linijo s pomocjo funkcij x in y
    this.Line = d3.svg.line().interpolate("basis")
        .x(function (d) {
          return this.X(d.CreatedDateTimeTimestamp);
        })
        .y(function (d) {
          return this.Y(d.Count);
        });
    // narisi linijo na podlagi objekta JSON z uporabo funkcije Line
    this.Graph.append("svg:path").attr("d",
        this.Line(this.Data)).attr("class", "timeline");
}
```

4.6 Primeri uporabe aplikacije v praksi

Aplikacija je bila prvič uporabljena za spremljanje predsedniških volitev 2012. Rezultati platforme, ki je spremljala vsakega kandidata posamezno, so bili interpretirani kot politični indeks. Politični indeks je predstavljal razliko med rezultati s pozitivnim in rezultati z negativnim sentimentom v določenem časovnem oknu. Na primer, če je za kandidata v zadnji uri zajetih tisoč rezultatov, od katerih je sedemsto pozitivnih in sto negativnih, je trenutni politični indeks kandidata šeststo.

Aplikacija je spremljala politični indeks skozi celotno predvolilno in volilno obdobje z enodnevnim in sedemdnevnim časovnim oknom za vse tri kandidate

in jih prikazovala na spletni strani, ki je bila javno dostopna.

Aplikacija je skozi celotno predvolilno obdobje in vsa predvolilna soočenja zaznavala najvišji politični indeks za Boruta Pahorja in je edina od vseh javnomnenjskih anket tako rekoč napovedala zmago Borutu Pahorju v prvem krogu, čeprav aplikacija ne služi kot javnomnenjska anketa in je kot take nismo niti oglaševali.

Kasneje je bila na enak način uporabljena za spremljanje bolgarskih volitev leta 2013.

Z aplikacijo smo spremljali tudi košarkarsko prvenstvo Eurobasket 2013, kjer smo jo uporabili za prikaz analize zadetkov, analize sentimenta in analize besednih oblakov, za iskalne kriterije, ki so se navezovali na samo prvenstvo in na reprezentance, ki so na njem nastopale.

Poglavje 5

Sklepne ugotovitve

Aplikacija za analizo in vizualizacijo podatkov iz družbenih omrežij, ki sem jo predstavil v tej diplomski nalogi, je dovolj hitra in učinkovita, da lahko obdela nekaj sto tisoč rezultatov na uro z le nekaj instancami. Število instanc pa je mogoče po potrebi povečati in s tem zadovoljiti tudi najzahtevnejše stranke.

Metode razvoja aplikacij v oblaku, ki sem jih predstavil na primeru aplikacije za analizo in vizalizacijo podatkov, se lahko uporabijo za poljubno aplikacijo, za katero je smiselno, da bodisi zaradi velikosti podatkov bodisi zaradi števila zahtevkov teče v oblaku. Federirane podatkovne baze, horizontalno in vertikalno skaliranje instanc ter porazdeljevanje bremena glede na prioriteto so načini, s katerimi je moč učinkovito izrabiti vire, ki nam jih nudi razvoj aplikacij v oblaku.

V prihodnje bi bila smiselna implementacija globinske analize, ki bi spremljala zgodovino objav uporabnikov, ki so prispevali rezultate za določene iskalne kriterije. S tem bi pridobili več podatkov o tem, kako neke akcije vplivajo na določene ciljne skupine, kar bi bilo zelo uporabno za oglaševalske namene.

Smiselna bi bila tudi implementacija spletnega vmesnika API v ločenem projektu s spletno avtentikacijo, ki bi vračala rezultate v objektih XML ali JSON. API bi bilo prav tako potrebno razširiti na vse analize, ki jih ponujamo preko spletne aplikacije.

Literatura

[1] T. Erl, Z. Mahmood, R. Puttini, Cloud Computing Concepts, Technology & Architecture. Westford: Pearson Education, 2013.

[2] P. Mell, T. Grance, The NIST Definition of Cloud Computing. Gaithersburg: National Institute of Standards and Technology 2011. Dostopno na: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[3] Spletna stran CloudTweaks, The 4 Primary Cloud Deployment Models. Dostopno na: http://www.cloudtweaks.com/2012/07/the-4-primary-cloud-deployment-models/

[4] Spletna stran CMS Wire, Cloud Service Models (IaaS, SaaS, PaaS) + How Microsoft Office 365, Azure Fit In. Dostopno na: http://www.cmswire.com/cms/information-management/cloud-service-models-iaas-saas-paas-how-microsoft-office-365-azure-fit-in-021672.php

[5] Spletna stran Windows Azure, Windows Azure Documentation. Dostopno na: http://www.windowsazure.com/en-us/documentation/

[6] Spletna stran Microsoft Developer Network, .NET Framework 4.5. Dostopno na: http://msdn.microsoft.com/en-us/library/vstudio/w0x726c2.aspx

[7] Spletna stran W3C, HTML & CSS. Dostopno na: http://www.w3.org/standards/webdesign/htmlcss

[8] Spletna stran Adobe, SVG Developer Center. Dostopno na: http://www.adobe.com/devnet/svg.html

[9] Spletna stran Mozilla Developer Network, JavaScript. Dostopno na: https://developer.mozilla.org/en-US/docs/Web/

[10] Spletna stran JQuery, JQuery. Dostopno na: http://jquery.com/

[11] Spletna stran Data-Driven Documents, D3.js. Dostopno na: http://d3js.org/

[12] H. Katwala, S. Shan, Study on Election Algorithm in Distributed System. Journal of Computer Engineering 2013. Dostopno na: http://www.iosrjournals.org/iosr-jce/papers/Vol7-Issue6/F0763439.pdf