Poizvedovanje po povezanih podatkih s porazdelitvijo dela med strežnikom in odjemalcem

Jan Robas in Dejan Lavbič

Jan Robas in **Dejan Lavbič**. 2018. **Poizvedovanje po povezanih podatkih s porazdelitvijo dela med strežnikom in odjemalcem**, Uporabna informatika **(UI)**, 26(2), str. 54 - 64.

Povzetek

Na spletu je veliko podatkov, ki so objavljeni na različnih mestih v različnih, nezdružljivih oblikah. Zaradi tega nad njimi ne moremo izvajati poizvedb. Iskanje odgovora na specifično vprašanje ponavadi terja iskanje po ključnih besedah, branje in tolmačenje vsebine. To lahko rešimo tako, da uporabimo povezane podatke. Problem pri tem je dostopnost in učinkovitost poizvedovanja. V tem članku predstavimo obstoječo rešitev, ki omogoča poizvedovanje po povezanih podatkih tako, da je delo porazdeljeno med strežnikom in odjemalcem. Poleg tega predstavimo obstoječo razširitev, ki poveča učinkovitost in ne nazadnje našo lastno razširitev, s katero poskušamo učinkovitost še izboljšati. Našo razširitev primerjamo z že obstoječo razširitvijo in z osnovno različico omenjene rešitve.

Ključne besede

Spletne tehnologije, povezani podatki, semantični splet, SPARQL, procesiranje na odjemalcu

Abstract

Querying Linked Data by Distributing Work between the Server and the Client

There is a lot of data on the web that is presented in various incompatible forms. Consequently, we can not simply query it. To get an answer to a specific question, we have to resort to searching by keywords and interpreting the results. We can solve this by using Linked Data. The problem with querying Linked Data is availability and performance. In this article, we present an existing solution that enables Linked Data querying by distributing the work between the server and the client. In addition, we present an existing extension that increases efficiency. We then present our own extension, by which we try to further improve the efficiency. We compare our extension with the existing one and with the basic version of the solution.

Keywords

Web technology, Linked Data, semantic web, SPARQL, client-side processing

1 Uvod

Uporabniki se vedno bolj zanašamo na pridobivanje podatkov preko svetovnega spleta. Na spletu se nahaja velika količina podatkov, ki je predstavljena v različnih, nezdružljivih oblikah. Enakovredni podatki so pogosto različno predstavljeni na večih mestih, kar pripelje do nekonsistence. Tudi spletni vmesniki, ki ponujajo podatke, so med sabo različni. Če želimo implementirati aplikacijo, ki uporablja te spletne vmesnike,

imamo lahko precej dela z usklajevanjem naših podatkov s podatki, ki jih dobimo preko njih. Kot omenjajo v članku (Heath and Bizer, 2011), moramo v naši aplikaciji za vsak spletni vmesnik posebej napisati kodo za odjemalca, kar zahteva znaten napor. Podatki, ki jih ponujajo spletni vmesniki, niso na voljo iskalnikom oziroma splošnim spletnim agentom. Za začetek reševanja teh problemov potrebujemo način objave podatkov, ki je prilagojen za internet. Prav tako je pomembno, da so podatki predstavljeni v strojno berljivi obliki, saj jih le tako lahko računalniško obdelujemo in po njih poizvedujemo. Eden od ustreznih načinov so povezani podatki[1], ki temeljijo na standardnih spletnih tehnologijah. V splošnem gre za tipizirane povezave med stvarmi iz različnih virov, pri čemer so tako stvari kot tipi povezav predstavljeni z URI-ji.

Po povezanih podatkih lahko poizvedujemo s pomočjo semantičnega poizvedovalnega jezika SPARQL. Pri tem se pojavijo problemi, povezani z dostopnostjo in učinkovitostjo poizvedovanja. V tem članku predstavimo obstoječ način objave povezanih podatkov, ki nam omogoča poizvedovanje s porazdelitvijo dela med strežnikom in odjemalcem. To rešitev nadgradimo s svojo razširitvijo in jo primerjamo z že obstoječo razširitvijo AMF[2].

V 2. poglavju bomo predstavili idejo povezanih podatkov. V 3. poglavju bomo predstavili delce povezanih podatkov in delce vzorcev trojčkov, ki omogočajo porazdelitev dela pri poizvedovanju. Nato bomo predstavili obstoječo razširitev AMF in našo razširitev. V 4. poglavju bomo našo razširitev evaluirali in jo primerjali z osnovno različico ter z razširitvijo AMF, potem pa sledi zaključek.

2 Povezani podatki

Pojem povezani podatki se nanaša na nabor dobrih praks objavljanja in povezovanja strukturiranih podatkov na spletu. V splošnem gre pri povezanih podatkih za tipizirane povezave med podatki iz različnih virov. Ti viri so lahko bodisi znotraj iste organizacije bodisi porazdeljeni med več različnih organizacij. Povezani podatki so objavljeni na spletu na tak način, da so strojno berljivi in jih je možno naknadno povezovati z ostalimi viri. Gre za povezovanje podatkov na podoben način, kot so na spletu povezani dokumenti HTML. Medtem ko so dokumenti HTML povezani z netipiziranimi povezavami, povezani podatki temeljijo na formatu RDF, kjer so povezave med objekti tipizirane. Rezultat je splet podatkov oziroma bolj točno splet objektov in pojmov, ki so predstavljeni s podatki na spletu (Bizer et al., 2011).

Osnovni principi povezanih podatkov se skladajo s principi in standardi svetovnega spleta in so naslednji (Berners-Lee, 2006):

1. Uporabi URI-je kot imena za stvari (tako za objekte kot pojme).
2. Uporabi HTTP URI-je tako, da lahko uporabniki do teh imen dostopajo (npr. preko spletnega brskalnika).
3. Ko uporabnik dostopa do URI-ja, ponudi koristne informacije z uporabo standardov (RDF, SPARQL).
4. Vključi povezave do drugih URI-jev, da lahko odkrijejo več stvari.

Povezani podatki temeljijo na temeljnih spletnih tehnologijah: URI in HTTP. Medtem ko z URL-ji naslavljamo posamezne dokumente in ostale vire na spletu, URI-ji omogočajo bolj splošno naslavljanje poljubnih entitet na svetu. Če se URI-ji začnejo s `http://`, potem jih lahko kot URL-je zahtevamo preko protokola HTTP in o njih pridobimo dodatne informacije. Čeprav URI-ji lahko predstavljajo pojme ali fizične stvari, ki jih ne moremo prenesti preko spleta, lahko o njih pridobimo opise, dodatne informacije in povezane podatke. Dve omenjeni tehnologiji sta dopolnjeni z RDF[3], ki definira podatkovni model v obliki grafa, v katerem povezujemo entitete s tipiziranimi povezavami. Podatki so zapisani v obliki trojčkov oblike subjekt, predikat, objekt. Tako subjekt kot objekt sta URI-ja, ki predstavljata vir oziroma v primeru objekta opcijski niz znakov (literal, ki lahko predstavlja bodisi niz znakov bodisi število). Predikat je tudi predstavljen z URI in pove, v kakšnem razmerju sta subjekt in objekt. Če imamo npr. predikat **`http://xmlns.com/foaf/0.1/knows`** in je prva oseba (subjekt) A s tem predikatom povezana z drugo osebo (objekt) B, si to razlagamo kot: oseba A pozna osebo B. V modelu RDF je to trojček (`A http://xmlns.com/foaf/0.1/knows B`), pri čemer sta A in B URI-ja, kjer vsak predstavlja svojo osebo. Dva vira (v našem primeru osebi A in B) sta lahko izvzeta

[1] angl. Linked Data
[2] angl. Approximate Membership Functions
[3] angl. Resource Description Framework

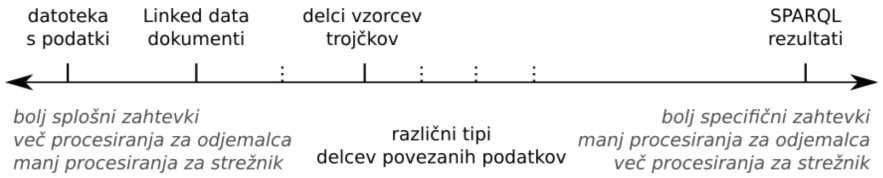

Figure 1: Vse spletne vmesnike, ki ponujajo trojčke RDF, si lahko predstavljamo kot ponudnike delcev vzorcev trojčkov

iz dveh različnih podatkovnih virov na spletu (npr. iz dveh družbenih omrežij). Ti podatki so med sabo združljivi. Trojčke RDF si zato lahko predstavljamo na podoben način, kot spletne povezave med stranmi HTML.

2.1 Osnovni pojmi

Vsak **trojček** vsebuje *subjekt*, *predikat* in *objekt*. Vsak od teh je predstavljen z URI-jem, pri čemer je objekt lahko, opcijsko, niz znakov.

Vzorec trojčka je podoben kot trojček, vendar za razliko od dejanskega trojčka, lahko namesto URI-jev in nizov znakov, vsebuje spremenljivke.

Rezultati vzorca trojčka so trojčki v bazi, ki jih lahko dobimo s preslikavo spremenljivk danega vzorca trojčka.

Vsebnostna poizvedba je poizvedba, ki preverja, če je nek trojček vsebovan v bazi.

Osnovni vzorec grafa BGP[4] je sestavljen iz večih vzorcev trojčka.

Poizvedba SPARQL je sestavljena iz enega ali več BGP-jev, ki so povezani z različnimi operatorji. Zaradi poenostavitve se v tem delu osredotočamo na poizvedbe, ki vsebujejo en BGP. Rezultati poizvedbe SPARQL so preslikave spremenljivk, ki vodijo do rešitve[5].

3 Delci povezanih podatkov

Članek (Buil-Aranda et al., 2013) govori o problemu dostopnosti oddaljenih točk SPARQL, saj je samo okoli tretjina le-teh dostopna 99 % časa. Ob koncu leta 2013 je bila povprečna dostopna točka SPARQL nedosegljiva več kot dan in pol vsak mesec. Poleg tega obstajajo velike razlike v učinkovitosti poizvedovanja ($3-4$ velikostnih redov). To pomeni, da ne moremo razviti zanesljive aplikacije, ki bi se zanašala na poizvedovanje preko tovrstnih dostopnih točk. Ena rešitev tega problema je, da uporabnikom ponudimo statične datoteke s podatki RDF. Problem ponujanja podatkov RDF v statičnih datotekah je, da so podatkovne zbirke lahko zelo velike, uporabnik pa morda potrebuje samo majhen del podatkov. Poleg tega mora imeti uporabnik nameščeno ustrezno programsko opremo. Omenjena načina dostopa do podatkov se da predstaviti z vidika delcev povezanih podatkov kot dve skrajnosti (glej sliko 1).

Osnovna ideja delcev povezanih podatkov[6] je, da so na dostopni točki objavljeni delci celotnega nabora povezanih podatkov (v obliki trojčkov), ki so razdeljeni po straneh. Pri tem lahko trojčke filtriramo z izbirnikom, ki ga lahko poljubno definiramo (Verborgh et al., 2014). V skladu z idejo povezanih podatkov so pri delcih povezanih podatkov prisotni metapodatki z dodatnimi informacijami o podatkih ter, kot jih imenuje omenjeni članek, kontrole, ki vsebujejo povezave do ostalih strani rezultatov in do ostalih delcev povezanih podatkov. Tako datoteko s podatki, kot dostopno točko SPARQL, lahko predstavimo iz vidika

[4]angl. Basic Graph Pattern
[5]angl. solution mappings
[6]angl. Linked Data Fragments

delcev povezanih podatkov. Kot vidimo na sliki 1, sta to skrajna pristopa, pri čemer datoteka s podatki predstavlja večjo obremenitev za odjemalca, dostopna točka SPARQL pa večjo obremenitev za strežnik. V tem članku bomo obravnavali eno od konkretnih implementacij delcev povezanih podatkov, delci vzorcev trojčkov[7], ki so opisani v članku (Verborgh et al., 2016). Namen delcev vzorcev trojčkov je izboljšati učinkovitost poizvedovanja po povezanih podatkih s porazdelitvijo obremenitve med strežnikom in odjemalcem.

3.1 Strežnik

Pri delcih vzorcev trojčkov je izbirnik vzorec trojčka. Strežnik ponuja rezultate vzorca trojčka, razdeljene po straneh. Delec vzorca trojčka, ki ga vrača strežnik, vsebuje:

- **podatke**: rezultate vzorca trojčka (na trenutni strani),
- **metapodatke**: približno število vseh trojčkov v rezultatih (po vseh straneh),
- **kontrole**: povezave (URL) do ostalih strani rezultatov podanega vzorca trojčka in do ostalih delcev vzorcev trojčkov (korenski URL do delca z vsemi trojčki v bazi).

Ker je izbirnik, namesto specifične poizvedbe SPARQL, v vsakem primeru le vzorec trojčka, se pogosto zgodi, da se pri večih različnih poizvedbah, tekom izvajanja, pojavi isti izbirnik. Če je vsaj en vzorec trojčka v BGP-ju skupen večim poizvedbam (kjer ime spremenljivke ni pomembno), pride do enakih zahtevkov. Zaradi tega je smiselno uporabiti posredniški predpomnilniški strežnik, ki posreduje podatke med strežnikom in odjemalcem. Ob tem si odgovore shranjuje v predpomnilnik, da jih potem ob enakih zahtevah lahko vrne odjemalcu brez obremenjevanja aplikacijskega strežnika.

3.2 Odjemalec

Odjemalec omogoča poizvedovanje po podatkih s pomočjo poizvedovalnega jezika SPARQL. Ker strežnik nudi dostop do delcev vzorcev trojčkov, mora odjemalec pridobiti ustrezne podatke s pomočjo le-teh. Zaželeno je, da odjemalec vsled izvedbe poizvedbe SPARQL naredi čim manj zahtevkov HTTP in prenese čim manj podatkov. Odjemalec uporablja metapodatke, konkretno število strani, za planiranje poizvedbe. BGP-ji vsebujejo enega ali več vzorcev trojčkov in so glavni gradniki poizvedb SPARQL. Ostali operatorji, ki se nahajajo med BGP-ji, se izvajajo na strani odjemalca in niso specifični za predstavljen način pridobivanja podatkov. Podpora teh je omejena in ni v obsegu članka. Odjemalec, razvit v okviru našega pristopa, je napisan v jeziku JavaScript in deluje v večini priljubljenih spletnih brskalnikih.

Odjemalec izvede BGP tako, da prenese prve strani delcev vseh nastopajočih vzorcev trojčkov. Tako v metapodatkih dobi število rezultatov za vsak vzorec trojčka. Nato izbere vzorec trojčka z najmanj rezultati in se začne po njem sprehajati. Za vsak trojček v rezultatih pripadajočo preslikavo (preslikavo spremenljivke v URI ali niz znakov) uporabi nad ostalimi vzorci trojčka v BGP-ju in nad novim BGP-jem rekurzivno izvede isti algoritem. Če vmes pride do delca brez rezultatov, potem preslikave, ki so vodile do tega, ne vodijo do rešitve in jih zavržemo. Kadar po preslikavi vzorec trojčka ne vsebuje spremenljivk, pride do t. i. vsebnostne poizvedbe. Gre za poizvedbe, s katerimi preverjamo vsebnost trojčka v bazi. Rezultat algoritma so vse preslikave, ki vodijo do rešitve.

Poizvedba 3.1. Poizvedba SPARQL, za katero odjemalec delcev vzorcev trojčkov naredi 397 vsebnostnih poizvedb.

```
PREFIX tags: <http://www.holygoat.co.uk/owl/redwood/0.1/tags/>
PREFIX tag: <http://dbtune.org/jamendo/tag/>
SELECT ?alb WHERE {
    ?alb tags:taggedWithTag tag:jazz. # 421
    ?alb tags:taggedWithTag tag:metal. # 397
}
```

[7] angl. triple pattern fragments

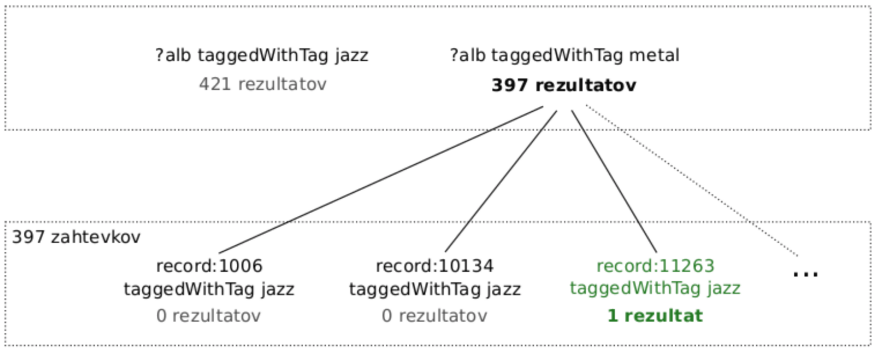

Figure 2: Prikaz izvedbe poizvedbe 3.1. Namesto celotnih URI-jev so zaradi berljivosti prikazani zadnji deli le-teh.

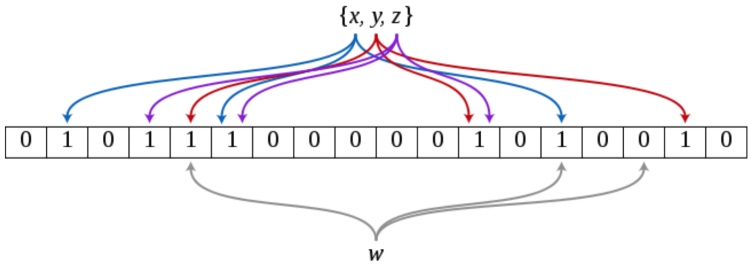

Figure 3: Bloomov filter

Za primer si poglejmo preprosto poizvedbo (glej poizvedbo 3.1) iskanja ustrezne preslikave za spremenljivko `?alb`, ki nastopa kot subjekt v obeh vzorcih trojčkov. V tem primeru iščemo subjekte, ki so označeni tako z oznako `tag:jazz` kot z oznako `tag:metal`. V poizvedbi nastopa en BGP, ki vsebuje vzorca trojčkov (`?alb tags:taggedWithTag tag:jazz`) in (`?alb tags:taggedWithTag tag:metal`). Iščemo torej subjekt, ki je prisoten v obeh delcih vzorcev trojčkov. Odjemalec izvede poizvedbo tako, da prenese prve strani obeh delcev in izbere vzorec trojčka, ki ima najmanj rezultatov. To je v tem primeru vzorec trojčka (`?alb tags:taggedWithTag tag:metal`) s 397 rezultati. Nato se začne po tem delcu sprehajati (po vseh straneh) in za vsak subjekt preveri, če se nahaja tudi v drugem pripadajočem delcu. Če se, je pripadajoča preslikava prisotna v rezultatih. Tako pride do 397 vsebnostnih poizvedb, kot je prikazano na sliki 2.

3.3 Obstoječa razširitev AMF

Razširitev AMF, kot jo opisuje članek (Vander Sande et al., 2015), na metapodatke delca vzorca trojčka doda verjetnostno podatkovno strukturo za množico, ki vsebuje vse trojčke, nastopajoče kot rezultati vzorca trojčka (ni razdelitve po straneh). Pri tem uporablja Bloomov filter, s pomočjo katerega lahko preverimo, če je možno, da se poljuben element nahaja v množici. Na vprašanje, če je element v množici, lahko odgovorimo z *mogoče* ali *ne*. Prazen Bloomov filter je niz m bitov, nastavljenih na 0. Vsak element se s pomočjo k zgoščevalnih funkcij preslika v bite v tem nizu. Pri dodajanju elementa v Bloomov filter se pripadajoči biti nastavijo na 1, pri preverjanju vsebnosti elementa pa se preveri, če so vsi pripadajoči biti nastavljeni na 1. Če so, potem je možno, da je element v množici, če pa niso, pa zagotovo vemo, da tega elementa v množici ni. Parametra m (velikost filtra) in k (število zgoščevalnih funkcij) sta lahko odvisna od velikosti množice, za katero gradimo Bloomov filter. S podano fiksno želeno verjetnostjo lažnih pozitivnih odgovorov velikost Bloomovega filtra linearno narašča s številom elementov v množici (Bonomi et al., 2006). Slika 3 prikazuje Bloomov filter, ki vsebuje elemente x, y in z, poizvedujemo pa po elementu w. Ker se w preslika v mesto, kjer je vrednost 0, vemo, da ga zagotovo ni v množici.

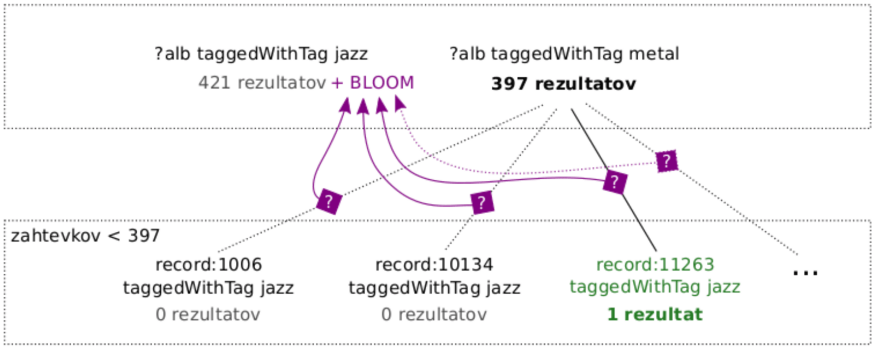

Figure 4: Prikaz izvedbe poizvedbe 3.1 z razširitvijo AMF.

Razširitev poleg Bloomovih filtrov omogoča tudi uporabo podatkovne strukture GCS[8], ki je prostorsko malo učinkovitejša, vendar porabi več procesorske moči (Putze et al., 2007). Medtem ko so rezultati razdeljeni po straneh, verjetnostna podatkovna struktura v metapodatkih obravnava celoten vzorec trojčka. Prednost te metode je, da zmanjša število vsebnostnih poizvedb (in posledično zahtevkov), saj nam ni potrebno preverjati vsebnosti trojčkov, za katere verjetnostna podatkovna struktura v metapodatkih ne izkazuje možnega vsebovanja v celotnem delcu vzorca trojčka.

Na sliki 4 je prikazana izvedba poizvedbe 3.1 preko delcev vzorcev trojčkov z opisano razširitvijo. Vijolične kvadratke si predstavljamo kot pogojne stavke, pri katerih vsebnostnih poizvedb ne izvedemo, če, glede na Bloomov filter v metapodatkih, ni možnosti vsebovanja danega trojčka. Parametra Bloomovega filtra m (dolžina bitnega polja) in k (število zgoščevalnih funkcij) se nastavita glede na število rezultatov v celotnem delcu vzorca trojčka tako, da imamo fiksno verjetnost napake oziroma lažno pozitivnega odgovora. Privzeta vrednost je $0,001$ oziroma $0,1\%$, s katero, glede na meritve, dobimo dobre rezultate.

3.4 Naš predlog izboljšave

Razširitev AMF zmanjša število vsebnostnih poizvedb, vendar lahko pri tem močno poveča velikost delcev. Naša ideja je, da v metapodatke dodamo trojčke, povezane z rezultati na prikazani strani delca vzorca trojčka, ki jih odjemalec prav tako lahko izkoristi za zmanjšanje števila vsebnostnih poizvedb. Pri tem se osredotočimo na izboljšanje v smislu količine prenesenih podatkov.

Tako kot razširitev AMF, tudi naša razširitev uporablja verjetnostno podatkovno strukturo, bolj specifično Bloomove filtre, vendar pri naši razširitvi obravnavamo druge podatke. Naša razširitev na metapodatke delca vzorca trojčka doda Bloomov filter, ki vsebuje trojčke, povezane s subjekti in objekti trojčkov na prikazani strani. Katere trojčke pri tem dejansko vsebuje Bloomov filter, je odvisno od različice naše razširitve.

Bloomov filter pri naši različici na strani i vsebuje:

- vse trojčke, povezane z objekti trojčkov na strani i, če je spremenljivka vzorca trojčka objekt,
- vse trojčke, povezane s subjekti trojčkov na strani i, če je spremenljivka vzorca trojčka subjekt,
- 4 različice, če sta spremenljivki vzorca trojčka tako subjekt kot object:
  - filtra ne vključimo (različica `Jan 1`),
  - v filtru so vsi trojčki, povezani s subjekti trojčkov na strani i (različica `Jan 2`),
  - v filtru so vsi trojčki, povezani z objekti trojčkov na strani i (različica `Jan 3`),
  - v filtru so vsi trojčki, povezani s subjekti ali objekti trojčkov na strani i (različica `Jan 4`).

Ker je lahko, v odvisnosti od podatkov, velikost Bloomovega filtra zelo velika, se odločimo za implementacijo omejitve števila trojčkov v Bloomovem filtru. Ko je določena omejitev števila trojčkov prekoračena, filtra ne pošljemo. Ta omejitev je nastavljiva.

[8]angl. Golomb-coded sets

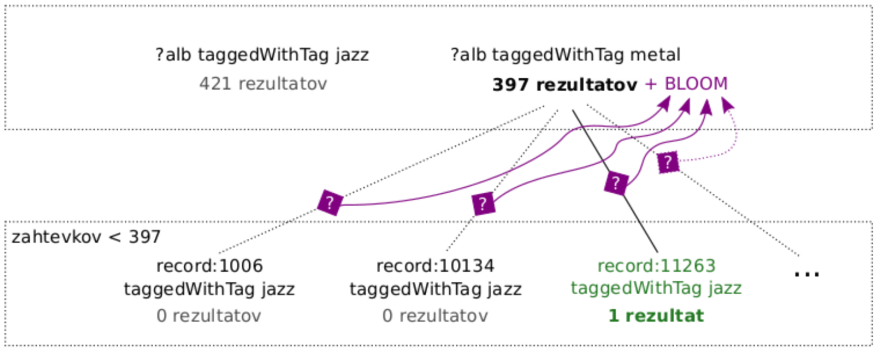

Figure 5: Prikaz izvedbe poizvedbe 3.1 z našo razširitvijo.

Dodatna možnost naše razširitve so različice, v tem članku imenovane **`Jan N Filter`**, pri čemer je N številka ene od zgoraj navedenih različic. Pri teh različicah v filter dodajamo le trojčke z določenimi predikati. Gre za omejitev trojčkov v filtru glede na predikat. V filtru so namreč uporabni le trojčki s predikati, ki nastopajo v vsebnostnih poizvedbah. S tem lahko dodatno zmanjšamo količino prenesenih podatkov, če vemo, kateri predikati se bodo v vsebnostnih poizvedbah pojavljali. Protokol za uskladitev seznama v filtru uporabljenih predikatov med strežnikom in odjemalcem ni določen; pri meritvah smo ta seznam ročno vnesli v programsko kodo strežnika in odjemalca, kot da gre za posebno različico, ki obravnava le določene predikate. Za splošno rabo bi bila potrebna sinhronizacija seznama predikatov pred vsakim izvajanjem poizvedbe, vendar bi bil zagotovo del teh podatkov proti podatkom, potrebnim za izvedbo celotne poizvedbe, zelo majhen. Če ta seznam ni popoln, še vedno lahko izvajamo poizvedbe, ki terjajo vsebnostne poizvedbe z drugimi predikati, vendar njihovo izvajanje ni optimalno. Te seznam bi lahko strežnik sestavil samodejno glede na vsebnostne poizvedbe, ki bi jih dobival.

Kot je razvidno iz slike 5, se uporaben Bloomov filter pri naši razširitvi nahaja na drugem delcu vzorca trojčka kot pri razširitvi AMF. Razlika je, da pri naši razširitvi Bloomov filter vsebuje trojčke, povezane z rezultati na prikazani strani, pri razširitvi AMF pa so v Bloomovem filtru vsi dejanski rezultati vzorca trojčka, ne glede na prikazano stran.

Z algoritmom 3.1 pridobimo Bloomov filter, ki ga priložimo delcu vzorca trojčka. Prikazan algoritem je uporabljen v različici **`Jan 2`**, saj v primeru, da sta tako objekt kot subjekt spremenljivki, v filter dodamo trojčke, povezane s subjekti rezultatov.

tp // vzorec trojcka
stran // prikazana stran rezultatov vzorca trojcka tp
$\boldsymbol{p_{napaka}}$ // verjetnost napake za Bloomov filter

VrniFilter(stran, tp) {
  if (tp["predikat"] je spremenljivka) {
    // vrni prazen filter
  } **else if** (tp["subjekt"] je spremenljivka || tp["objekt"] je spremenljivka) {
    del = "subjekt"
  } **else if** (tp["objekt"] je spremenljivka) {
    del = "objekt"
  }
  rezultati = prazna mnozica
  za vsak trojcek na strani {
    q_1 = poizvedba : trojcek[del] ?p ?o
    q_2 = poizvedba : ?s ?p trojcek[del]
    rezultati = rezultati $\cup$ rezultati $q_1 \cup$ rezultati q_2
  }
  if (|rezultati| > omejitev) {
    // vrni prazen filter
  }
  // nastavi optimalna parametra m in k
  $m = \frac{\lceil -|\text{rezultati}| \cdot p_{napaka} \rceil}{ln(2)^2}$
  $k = \frac{m}{|\text{rezultati}| \cdot ln(2)}$
  bloom = nov Bloomov filter s parametroma m in k
  za vsak trojcek v rezultatih {
    bloom.dodaj(trojcek["subjekt"] + "|" + trojcek["predikat"] + "|" + trojcek["subjekt"])
  }
  // bloom tukaj predstavlja vrednost Bloomovega filtra v zapisu base64
  vrni m + "," + k + "," + bloom
}

Algoritem 3.1. Psevdokoda za izgradnjo našega filtra.

Naša razširitev je bolj podrobno opisana v magistrskem delu (Robas, 2016).

4 Evaluacija predlagane izboljšave

4.1 Metodologija

Za avtomatizirano izvajanje meritev smo uporabili BrowserMob Proxy[9] in Selenium WebDriver[10] z brskalnikom Chromium.

Pri ocenjevanju uspešnosti razširitve smo obravnavali naslednji metriki:

- **Število zahtevkov do strežnika**. Z našo razširitvijo želimo zmanjšati število vsebnostnih poizvedb. Vsaka vsebnostna poizvedba predstavlja zahtevek HTTP.
- **Količina prenesenih podatkov od strežnika proti odjemalcu**. Z zmanjšanjem števila vsebnostnih poizvedb se zmanjša količina prenesenih podatkov, vendar pa filter v metapodatkih poveča velikost delcev vzorcev trojčkov. Ker dandanes za dostop do informacij pogosto uporabjamo mobilne naprave, je ta metrika pomembna. Pri naši razširitvi smo se osredotočili na optimizacijo iz vidika te metrike.

Računalnik, na katerem je potekalo testiranje, ima `CPE Intel Core i7 6700`, ki teče s frekvenco `3,4 GHz`, in `16 GiB` delovnega pomnilnika. Testiranje je potekalo v operacijskem sistemu `Ubuntu 16.04 LTS`. Pri obeh metrikah velja, da je manjša vrednost boljša, saj obe predstavljata vir, ki ga hočemo pri izvajanju poizvedb čim manj izrabiti.

Za testiranje smo uporabili Waterloo SPARQL Diversity Test Suite (Aluc et al., 2014) z zbirko, ki vsebuje 10 milijonov trojčkov. Različne razširitve in osnovno različico brez razširitev smo testirali s 340 poizvedbami SPARQL. Te poizvedbe so generirane z WatDiv tako, da je za vsako od predlog C_1, F_1, F_2, F_3, F_4, F_5, L_1, L_2, L_3, L_4, L_5, S_1, S_2, S_3, S_4, S_5 in S_6 generiranih 20 poizvedb. Vsaka od teh predlog predstavlja določeno obliko poizvedb, naključno pa se spreminjajo določeni URI-ji. Enkrat npr. iščemo igralce, ki se nahajajo v mestu A, drugič pa igralce, ki se nahajajo v mestu B, sama oblika poizvedbe, če bi jo predstavili z grafom, pa ostaja enaka. Izvedba poizvedb, ustvarjenih iz predlog F_1, L_2, L_5, S_2, S_3, S_4, S_5 in S_6, terja vsebnostne poizvedbe. Pri izvajanju teh poizvedb z našo razširitvijo pričakujemo manjše število zahtevkov, manjšo količino prenesenih podatkov in (posledično) krajši čas izvajanja v primerjavi z osnovno različico. Pri ostalih poizvedbah pričakujemo večjo količino prenesenih podatkov zaradi dodanega Bloomovega filtra. Verjetnost napake za Bloomove filtre je bila pri vseh razširitvah nastavljena na $0,1\%$. Na vseh stolpičnih grafih je prikazano povprečje obravnavane metrike pri izvedbi vseh poizvedb oziroma podmnožice le-teh. Pri intervalih zaupanja smo uporabili 5% stopnjo tveganja.

Pri razširitvah Jan N Filter smo v filter dodali trojčke, katerih predikati nastopajo v vsebnostnih poizvedbah katerekoli od vseh 340 poizvedb, ki nastopajo v testu. Te predikate smo dobili tako, da smo vse poizvedbe pognali z osnovno različico in si ob vsaki vsebnostni poizvedbi shranili njen predikat. Dobili smo 7 različnih predikatov, ki smo jih pred testom sinhronizirali med strežnikom in odjemalcem. S protokolom, kjer bi potrebovali uskladitev enkrat na izvedbo poizvedbe, bi z obravnavanimi podatki in poizvedbami potrebovali okoli 1 KiB dodatnih podatkov na poizvedbo. Pri naših testnih podatkih in poizvedbah je povprečna količina prenesenih podatkov pri izvajanju poizvedbe z razširitvijo `Jan 2 Filter` okoli $3,5$ MiB, s tem da je mediana okoli 1 MiB, torej bi bila razlika sorazmerno majhna. Pri vseh različicah `Jan N Filter` je omejitev števila trojčkov v filtru nastavljena na 20 tisoč.

4.2 Rezultati meritev

4.2.1 Količina prenesenih podatkov

Na sliki 6 je prikazan graf primerjave različic glede na povprečno količino prenesenih podatkov pri izvedbi vseh poizvedb. Najmanj podatkov prenese različica z razširitvijo `Jan 2`. V nadaljevanju podrobnejše analize našega pristopa bomo obravnavali razširitvi `Jan 2` in `Jan 4`.

[9] https://github.com/lightbody/browsermobproxy
[10] http://www.seleniumhq.org

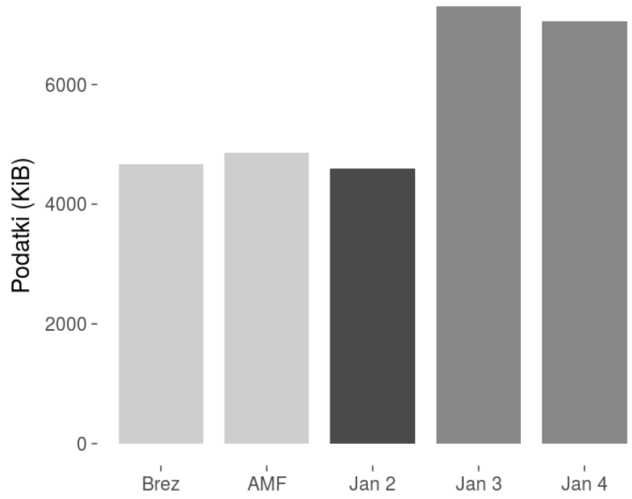

Figure 6: Primerjava povprečne količine prenesenih podatkov pri izvedbi vseh 340 poizvedb SPARQL z različnimi razširitvami.

Količino prenesenih podatkov smo pri naši razširitvi poskusili zmanjšati z omejitvijo števila trojčkov v filtru. Ker je pri nas filter odvisen od trojčkov na prikazani strani, za poljuben vzorec trojčka prekoračitev na eni strani ne implicira prekoračitve na drugih straneh. Številka v oklepaju poleg oznake predstavlja omenjeno omejitev. Črka k je oznaka za *kilo* ($10k$ npr. predstavlja 10 tisoč). Izmerili smo količino prenesenih podatkov ob uporabi razširitve `Jan 2` pri omejitvah 10 tisoč, 20 tisoč in 30 tisoč. Kot je razvidno iz grafa na sliki 7, se izkaže, da je količina prenesenih podatkov najmanjša pri omejitvi 20 tisoč, zato smo se odločili za nastavitev omejitve na to vrednost.

Primerjava povprečne količine prenesenih podatkov pri izvedbi poizvedb, ki terjajo vsebnostne poizvedbe, je prikazana na sliki 8. Iz grafa je razvidno, da, izvzemši rezultate različic `Jan N Filter`, od naših različic najmanj podatkov prenese `Jan 2`. Kar se tiče prenosa podatkov, je najbolj učinkovita različica `Jan 4 Filter`, vendar moramo za učinkovito uporabo te različice predhodno vedeti, kateri predikati bodo nastopali v vsebnostnih poizvedbah.

Na sliki 9 vidimo primerjavo povprečne količine prenesenih podatkov pri izvedbi poizvedb, ki ne terjajo vsebnostnih poizvedb. Učinkovitost je pri teh poizvedbah ob uporabi vseh opisanih razširitev slabša kot pri osnovni različici, saj vse razširitve z dodajanjem metapodatkov poskušajo zmanjšati število pri izvedbi teh poizvedb neobstoječih vsebnostnih poizvedb. Iz grafa je razvidno, da naše različice pri teh poizvedbah dodajo manj podatkov kor različica z razširitvijo AMF.

Povprečna količina prenesenih podatkov pri izvedbi vseh poizvedb je prikazana na sliki 10. Na grafu lahko vidimo, da najmanj podatkov prenesemo pri uporabi različice `Jan 4 Filter`, vendar moramo za take rezultate predhodno vedeti, kateri predikati bodo nastopali v vsebnostnih poizvedbah.

4.2.2 Število zahtevkov

Na sliki 11 je prikazana primerjava razširitev glede na povprečno število zahtevkov pri izvedbi vseh 340 poizvedb. Vidimo lahko, da sta razširitvi `Jan 2` in `Jan 4` malo manj učinkoviti kot razširitvi AMF in `Jan 4 Filter`. Ker z vsemi razširitvami, navedenimi v tem članku, zmanjšujemo število vsebnostnih poizvedb, je število zahtevkov pri poizvedbah, ki ne terjajo vsebnostnih poizvedb enako kot pri osnovni različici. Zaradi tega je razlika v številu zahtevkov bolj opazna na sliki 12. Pri razširitvi `Jan 4`, ki je najbolj splošna, je pričakovano podobno zmanjšanje števila zahtevkov kot pri razširitvi AMF. Izkaže se, da ob uporabi razširitve `Jan 4` pri poizvedbah iz sklopa S_4 prihaja do prekoračitve števila trojčkov v filtru in posledično filter ni poslan. Na sliki 13 vidimo primerjavo razširitev pri izvajanju poizvedb iz sklopa S_4. Ob omejitvi trojčkov v filtru glede na predikat (`Jan 4 Filter`) se filter splača poslati in je učinkovitost podobna kot pri razširitvi

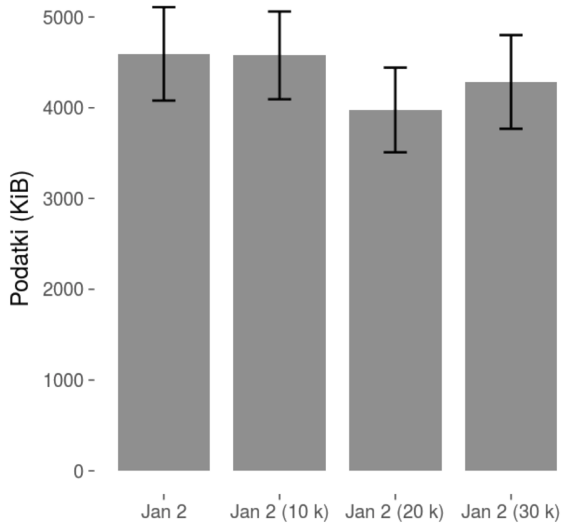

Figure 7: Primerjava povprečne količine prenesenih podatkov pri izvedbi vseh 340 poizvedb SPARQL z razširitvijo `Jan 2` in uporabo različnih omejitev števila trojčkov v filtru.

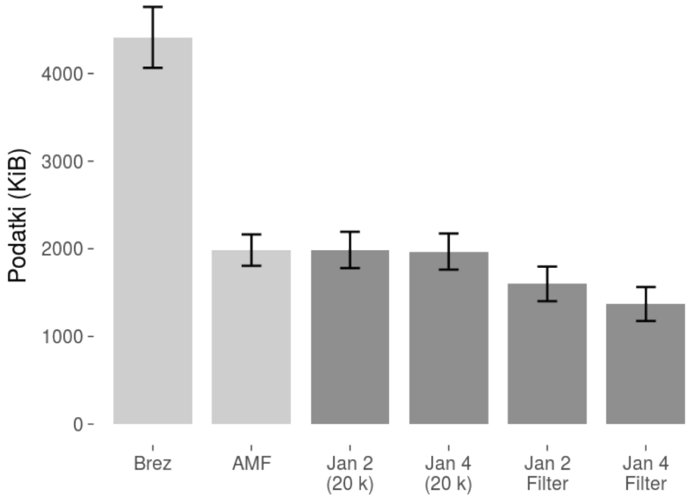

Figure 8: Primerjava povprečne količine prenesenih podatkov pri izvedbi sklopov poizvedb, ki terjajo vsebnostne poizvedbe (F_1, L_2, L_5, S_2, S_3, S_4, S_5, S_6), z različnimi razširitvami.

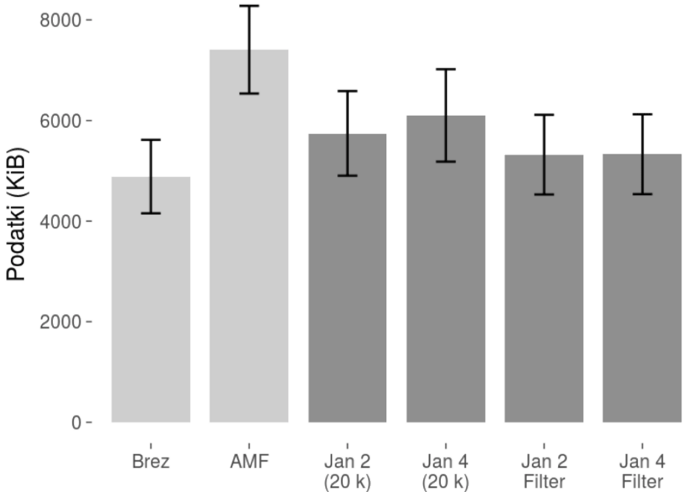

Figure 9: Primerjava povprečne količine prenesenih podatkov pri izvedbi sklopov poizvedb, ki ne terjajo vsebnostnih poizvedb (C_1, F_2, F_3, F_4, F_5, L_1, L_3, L_4, S_1), z različnimi razširitvami.

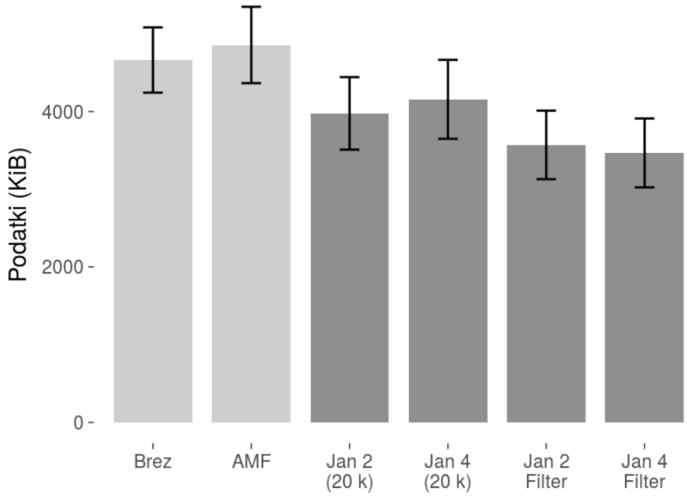

Figure 10: Primerjava povprečne količine prenesenih podatkov pri izvedbi vseh 340 poizvedb z različnimi razširitvami, s tem da pri naših razširitvah uporabljamo omejitev števila trojčkov v filtru.

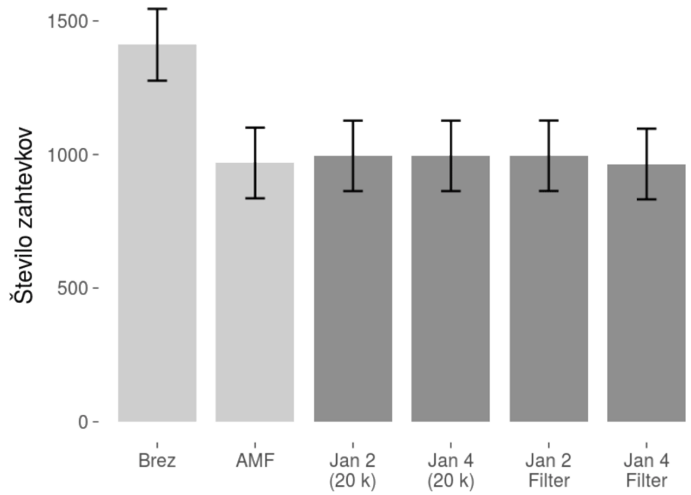

Figure 11: Primerjava povprečnega števila zahtevkov pri izvedbi vseh 340 poizvedb.

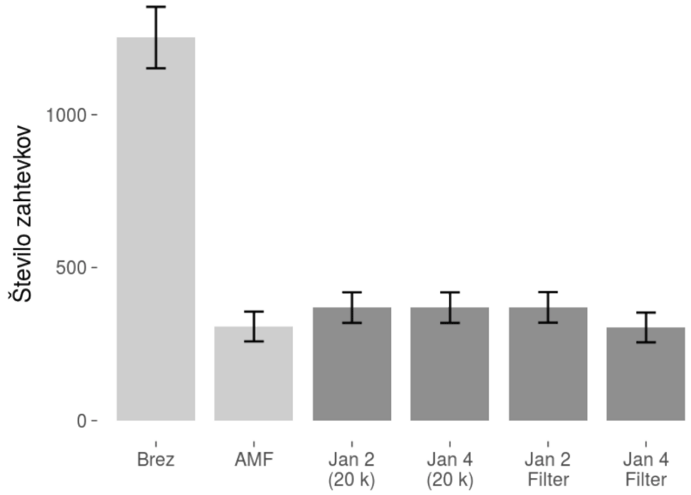

Figure 12: Primerjava povprečnega števila zahtevkov pri izvedbi sklopov poizvedb, ki terjajo vsebnostne poizvedbe (F_1, L_2, L_5, S_2, S_3, S_4, S_5, S_6), z različnimi razširitvami.

AMF. Razširitvi `Jan 2` in `Jan 4` sta brez uporabe filtra predikatov pri tem sklopu poizvedb manj učinkoviti kot razširitev AMF.

5 Zaključek

Čeprav je nevtralnost interneta zaradi različnih interesov pogosto pod vprašanjem (Ganley and Allgrove, 2006), nam spletne tehnologije omogočajo prosto prejemanje in objavljanje vsebine. Velikokrat pa je problem, kako to vsebino ponuditi na tak način, da jo bo kdo lahko našel. Rešitev za to so lahko povezani podatki, vendar je pri tem potrebno razviti način, kako po veliki količini povezanih podatkov iskati. Delci vzorcev trojčkov predstavljajo splošen način iskanja po povezanih podatkih s porazdelitvijo bremena med strežnikom in odjemalcem. Poleg tega so zasnovani tako, da so razširljivi. V tem članku smo razvili svojo razširitev in jo primerjali z že obstoječo razširitvijo AMF. Ugotovili smo, da naša razširitev, glede na test WatDiv, z ustrezno nastavitvijo parametrov dosega dobre rezultate. Pri našem testu različica z našo razširitvijo prenese manj podatkov kot različica z razširitvijo AMF in različica brez razširitve.

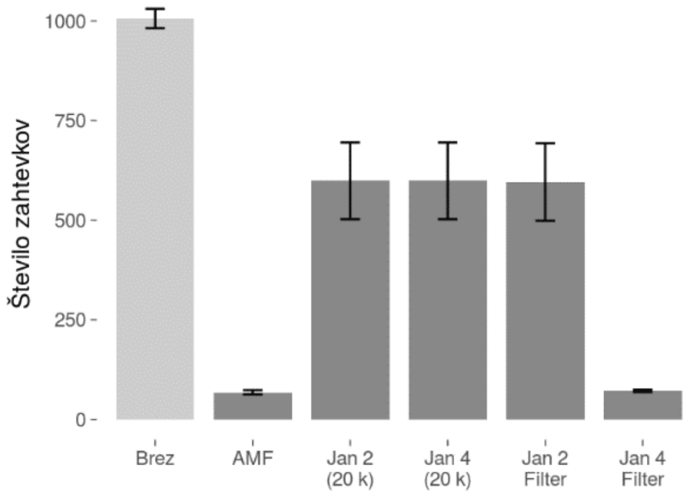

Figure 13: Primerjava povprečnega števila zahtevkov pri izvedbi poizvedb v sklopu S_4.

V nadaljevanju bi bilo potrebno za metode `Jan N Filter` določiti protokol za uskladitev seznama uporabljenih predikatov med strežnikom in odjemalcem. Lahko bi implementirali tudi algoritem, ki bi ta seznam na strani strežnika sproti prilagajal in bi tako omogočil bolj optimalno izvajanje poizvedb brez predhodne konfiguracije. Dobre rezultate bi lahko dobili tudi z nepopolnim seznamom predikatov.

Glede na pridobljene rezultate menimo, da je naša razširitev praktično uporabna. Še posebej dobro se obnese v primerih, ko nad podatki delamo poizvedbe, pri katerih se pogosto pojavljajo enaki predikati. Če uporabljamo razširitev `Jan 4 Filter`, lahko iskanje optimiziramo za najpogostejše poizvedbe in tako dosežemo še boljšo učinkovitost. Učinkovitost vsake razširitve pa je seveda odvisna tako od podatkov kot od poizvedb, ki jih izvajamo.

References

Aluc, G., Hartig, O., Ozsu, M. T., and Daudjee, K. (2014). Diversified Stress Testing of RDF Data Management Systems. In Mika, P and Tudorache, T and Bernstein, A and Welty, C and Knoblock, C and Vrandecic, D and Groth, P and Noy, N and Janowicz, K and Goble, C, editor, *Semantic Web - ISWC 2014, PT I*, volume 8796 of *Lecture Notes in Computer Science*, pages 197–212, Riva del Garda, Italy. Elsevier; Yahoo Labs; Telefonica; Parlance Project; Google; IBM Res; Microsoft Res; LinkedUp Project; Open PHACTS Fdn; Systap; Semant Valley; OpenLink Software; Poolparty; Fluid Operat; News Reader Project; Synerscope; Okkam; SICRaS; Bpeng; Centro Ric GPI; Expert Syst; Informatica Trentina s p a; CNR, Inst Cognit Sci & Technologies; iMinds; Videolectures; Fondazione Bruno Kessler; Semant Web Sci Assoc.

Berners-Lee, T. (2006). Linked Data - Design Issues.

Bizer, C., Heath, T., and Berners-Lee, T. (2011). Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI Global.

Bonomi, F., Mitzenmacher, M., Panigrahy, R., Singh, S., and Varghese, G. (2006). An improved construction for counting bloom filters. In Azar, Y and Erlebach, T, editor, *ALGORITHMS - ESA 2006, PROCEEDINGS*, volume 4168 of *Lecture Notes in Computer Science*, pages 684–695. European Assoc Theoret Comp Sci.

Buil-Aranda, C., Hogan, A., Umbrich, J., and Vandenbussche, P.-Y. (2013). Sparql web-querying infrastructure: Ready for action? In *International Semantic Web Conference*, pages 277–293. Springer.

Ganley, P. and Allgrove, B. (2006). Net neutrality: A user's guide. *Computer Law & Security Review*, 22(6):454–463.

Heath, T. and Bizer, C. (2011). Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136.

Putze, F., Sanders, P., and Singler, J. (2007). Cache-, hash-and space-efficient bloom filters. In *International Workshop on Experimental and Efficient Algorithms*, pages 108–121. Springer.

Robas, J. (2016). Učinkovito poizvedovanje po povezanih podatkih s porazdelitvijo obremenitve. Master's thesis, University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia.

Vander Sande, M., Verborgh, R., Van Herwegen, J., Mannens, E., and Van de Walle, R. (2015). Opportunistic Linked Data Querying Through Approximate Membership Metadata. In Arenas, M and Corcho, O and Simperl, E and Strohmaier, M and DAquin, M and Srinivas, K and Groth, P and Dumontier, M and Heflin, J and Thirunarayan, K and Staab, S, editor, *SEMANTIC WEB - ISWC 2015, PT I*, volume 9366 of *Lecture Notes in Computer Science*, pages 92–110. Elsevier; Fujitsu; Google; iMinds; Ontotext; Systap; Yahoo Labs; Franz Inc; IBM Res; Oracle; Metaphacts; Blazegraph.

Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., and Van de Walle, R. (2014). Querying datasets on the web with high availability. In *International Semantic Web Conference*, pages 180–196. Springer.

Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., and Colpaert, P. (2016). Triple Pattern Fragments: A low-cost knowledge graph interface for the Web. *Journal of Web Semantics*, 37-38:184–206.