VICOS EYE – SPELTNA STORITEV ZA KATEGORIZACIJO VIZUALNIH OBJEKTOV

Domen Tabernik, Luka Čehovin, Matej Kristan, Marko Boben, Aleš Leonardis

Laboratorij za umetne vizualne spoznavne sisteme
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani
E-pošta: domen.tabernik@fri.uni-lj.si

POVZETEK: *V članku predstavimo arhitekturo sistema za spletno storitev, ki omogoča poganjanje naprednih algoritmov računalniškega vida porazdeljenih preko večjega števila računalnikov. Arhitekturno sistem ločimo na učenje, tokovno procesiranje v realnem času in uporabniški vmesnik za spletno storitev. Učenje implementiramo v domeni MapReduce s pomočjo Hadoop poslov, medtem ko implementiramo realno-časovno procesiranje kot aplikacijo na sistemu Storm. Kot spletni vmesnik za končnega uporabnika dodatno implementiramo tudi spletno stran in Android aplikacijo. Sistem testiramo na naši gruči računalnikov in pokažemo, da se lahko slike iz podatkovne zbirke Caltech-101 naučimo v 40 minutah, medtem ko lahko tokovno procesiranje v realnem času obdela posamezno vhodno zahtevo v manj kot dveh sekundah.*

1. UVOD

Nenehen dvig procesorske moči je v preteklih letih na področju računalniškega vida omogočil pojav spletnih storitev, ki jih v preteklosti ni bilo mogoče implementirati predvsem zaradi prevelike računske intenzivnosti algoritmov iz tega področja. Nekatere izmed novih storitev so TinEye[1], Macroglossa[2], Google Image Search[3] in Google Goggles[4]. Vse omenjene storitve delujejo dobro predvsem na slikah, ki so jih predhodno že opazili na spletu, vendar odpovejo pri slikah, ki jih sistem še ni videl. Na primer, TinEye ne izvaja popolnoma nobenega prepoznavanja objektov, medtem ko pri vnosu slike posnete z digitalnim fotoaparatom na Google Image Search le-ta ne vrne nobenega relevantnega zadetka, če se na sliki nahaja določen splošen objekt kot sta na primer stol ali pa skodelica za kavo (slika 1).

[1] http://www.tineye.com
[2] http://www.macroglossa.com
[3] http://images.google.com
[4] http://www.google.com/mobile/goggles

Slika 1: Primer vrnjenih rezultatov iz Google Image Search za neznane slike.

V našem članku zato predstavimo arhitekturo sistema, ki podobno kot zgoraj omenjeni sistemi, deluje kot spletna storitev in obenem omogoča procesiranje vhodnih slik z bolj naprednimi algoritmi računalniškega vida. Pri tem je glavni cilj članka predstavitev arhitekture takšnega sistema, ki je zmožen omenjene algoritme poganjati porazdeljeno na večjem skupku računalnikov in s tem omogočiti enostavno prilagoditev storitve ob povečevanju vhodnega prometa. V splošnem je sistem zasnovan za poganjanje poljubnih algoritmov računalniškega vida, vendar se v našem primeru osredotočimo le na določen algoritem. Z našo storitvijo želimo zagotavljati kategorizacijo objektov, tako da uporabnik vnese sliko oz. izbrano regijo slike, ter jo nato naš sistem obdela, s tem da ustvari ustrezno značilko HoC [5] in jo klasificira s pomočjo metode podpornih vektorjev (SVM) [1] v eno izmed v naprej naučenih kategorij. Informacijo o kategoriji se nato vrne uporabniku, ki lahko to informacijo uporabi za različne namene, odvisno od želenega namena aplikacije. Na primer uporabnik informacije je lahko robotski sistem, ki želi identifikacijo objekta, ki ga vidi pred sabo, ali pa je to lahko uporabnik, ki želi prepoznati določeno vrsto živali, rastline in podobno.

2. ZAHTEVE SISTEMA

Pri implementaciji sistema moramo upoštevati sledeče zahteve:

- nudenje spletne storitve za bolj napredne algoritme računalniškega vida
- zmožnost poganjanja algoritmov preko večjega skupka računalnikov v porazdeljenem načinu
- zmožnost hitre obdelave po več sto zahtevkov na sekundo

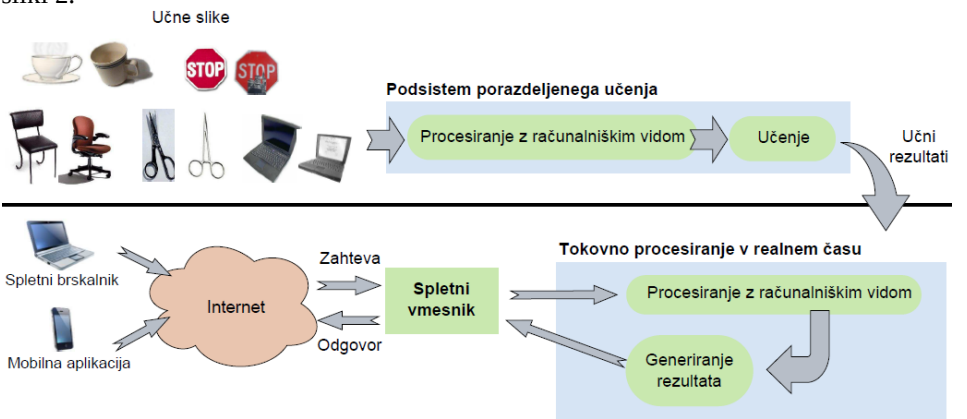
Pri prvi zahtevi moramo zagotoviti, da bo naš sistem služil kot podpora za spletno storitev, pri kateri se lahko promet konstantno spreminja. Zaradi tega moramo zagotoviti učinkovito fleksibilnost sistema, da bo imel zmožnost nenehnega prilagajanja prometu, ki se s časom lahko močno poveča. To zagotovimo z implementacijo algoritma v načinu, ki omogoča porazdelitev procesiranja preko večjega števila računalnikov. Porazdeljena implementacija mora biti dovolj fleksibilna, da lahko z enostavnim dodajanjem nove procesorske moči zagotovimo učinkovito rabo celotne računske zmogljivosti in s tem omogočimo večjo prepustnost sistema. To dosežemo z implementacijo algoritma v sistemih Hadoop [4] in Storm [6], ki sta specifično namenjena za porazdeljeno procesiranje in zelo enostavno omogočata povečanje prepustnosti s preprostim dodajanjem nove procesorske moči.

Za učinkovito zagotavljanje spletne storitve moramo prav tako poskrbeti za procesiranje večjega števila zahtevkov sočasno, pri čemer morajo biti vsi zahtevki obdelani v zelo kratkem času. V splošnem se za spletne strani priporoča čas TWT (ang. *tolerable waiting*

time) v rangu dveh sekund [1], vendar pa lahko le-ta varira glede na specifično aplikacijo. V našem primeri bi z zagotovljenim odzivnim časom do 5 sekund lahko omogočili uporabo sistema v različnih aplikacijah.

3. ARHITEKTURA

Za zagotavljanje zgornjih zahtev sistem arhitekturno razdelimo v dva glavna podsistema: *porazdeljeno učenje* in *tokovno procesiranje v realnem času*. Pri prvem podsistemu implementiramo učinkovito procesiranje večjega števila slik v paketnem načinu in ustrezno učenje na podlagi obdelanih slik. Medtem ko drugi podsistem priklopimo direktno na spletno storitev in lahko tako zagotavlja hitro procesiranje in streženje vseh vhodnih zahtev iz spleta. Kot tretji podsistem velja omeniti še uporabniški vmesnik, ki uporabnikom zagotavlja dostop do storitev sistema, in tako deluje kot posrednik med uporabnikom in sistemom za procesiranje zahtev. Spletni vmesnik smo implementirali kot preprosto spletno stran ter kot Android aplikacijo. Celotna arhitektura je prikazana na sliki 2.



Slika 2: Dvostopenjska arhitektura sistema, ki zagotavlja storitev računalniškega vida.

4. PODSISTEM ZA PORAZDELJENO UČENJE

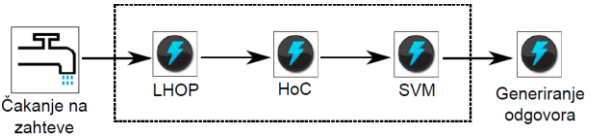
V tem poglavju opisani podsistem je zadolžen za učenje knjižnic ali modelov, ki so nato potrebni pri izvajanju podsistema za tokovno procesiranje v realnem času. V našem primeru kategorizacije vizualnih objektov s pomočjo značilke HoC in metode podpornih vektorjev (SVM) to pomeni, da moramo vse učne slike v prvem koraku obdelati z metodo LHOP [7] in ustvariti ustrezne značilke HoC. V naslednjem koraku pa te značilke nato vstopijo kot učne vrednosti metode podpornih vektorjev. S to metodo je potrebno naučiti ločen model za vsako od učenih kategorij.

Zgoraj opisan postopek realiziramo v domeni MapReduce [2], kar nam omogoči, da lahko sistem zelo enostavno prilagodimo za povečano množico vhodnih podatkov, oz. ogromno količino slik v našem primeru. Za implementacijo MapReduce smo vzeli odprto-kodni sistem Hadoop [4], ki je spisan v Javi. Glavne značilnosti tega sistema so

zmožnost procesiranja ogromne količine podatkov (več deset Terabyte-ov podatkov) in relativno enostavna skaliranost na gruče računalnikov z 1000 ali več vozlišči.
Za učinkovito implementacijo v MapReduce domeni je potrebno identificirati vhodne elemente, ki se lahko procesirajo neodvisno od ostalih elementov, in jih lahko Hadoop sistem samodejno razporedi po razpoložljivih procesorskih enotah. V primeru prvega koraka kategorizacije vizualnih objektov so ti vhodni elementi kar slike, saj lahko za vsako sliko ustvarimo značilko HoC povsem neodvisno od ostalih slik. Paralelizacija na tem nivoju je smiselna tudi zaradi ogromnega števila vhodnih slik (od 1000 do 100000 slik), ki jih potrebujemo za učenje kategorij. Ta korak implementiramo kot en Hadoop posel, ki se tako izvede za vse učne slike naenkrat, nato pa se jih shrani na porazdeljen datotečni sistem (HDFS). Izhod tega posla služi kot vhod naslednjem poslu, ki implementira učenje vseh kategorij z metodo podpornih vektorjev. V koraku strojnega učenja za vhodni element paralelizacije ne izberemo posamezne slike, ampak izberemo posamično kategorijo, saj je potrebno za učenje ene kategorijo uporabiti vse učne slike (bodisi za pozitivne primere bodisi za negativne primere). V našem primeru smo za implementacijo SVM uporabili LIBSVM [1], vendar je Hadoop posel napisan tako splošno, da bi bilo mogoče metodo zamenjati s poljubno metodo strojnega učenja.
Pri obstoječi implementaciji Hadoop posla za učenje z metodo podpornih vektorjev smo lahko večjo gručo računalnikov izkoristili tudi za optimizacijo parametrov SVM metode (ang. *parameter grid optimization*). To smo storili enostavno s podvajanjem učnih kategorij, pri čemer smo za vsak element paralelizacije izbrali poleg kategorije še ustrezno vrednost parametra, ki ga optimiziramo. Metoda Reduce v postopku MapReduce je nato služila za izbiro najbolj optimalnega naučenega modela za vsako kategorijo posebej.

5. PODSISTEM ZA TOKOVNO PROCESIRANJE V REALNEM ČASU

V tem podsistemu je potrebno vsako zahtevo, ki pride iz interneta obdelati z metodo LHOP, ustvariti ustrezno značilko HoC ter značilko klasificirati v eno od predhodno naučenih kategorij. Ta postopek je zaradi zahtev po spletni storitvi potrebno narediti v manj kot 5 sekundah, kar izključi uporabo Hadoop sistema za porazdeljeno procesiranje tega problema. Hadoop je namenjen paketnemu procesiranju ogromne količine podatkov naenkrat, kar pomeni, da je potrebno vse vhodne podatke pripraviti v naprej. Zaradi nepredvidljivega prihajanja zahtev v sistem pa to v tem podsistemu ni mogoče zagotoviti.
V tem primeru lahko učinkovito paralelizacijo izvedemo z implementacijo aplikacije v sistemu Storm [6]. Slednji je namenjen za porazdeljeno procesiranje v realnem času in omogoča izredno hiter odziv na posamezno zahtevo.
Aplikacijo v Storm sistemu definiramo s t.i. topologijo (usmerjen graf), ki določa posamezne procesne elemente in tok podatkov, ki se pretaka med njimi. Primer topologije za našo aplikacijo je prikazan na sliki 3.



Slika 3: Storm topologija za kategorizacijo vizualnih objektov z uporabo značilke HoC in metode podpornih vektorjev.

Naša topologija je sestavljena iz enega izvora (ang. *spout*), ki čaka na vhodne zahteve ter jih nato posreduje naslednjim elementom v procesiranje. Naslednji procesni element (ang. *bolt*) obdela sliko z LHOP knjižnico, in podatke pošlje v naslednji element. Ta ustvari značilko HoC in jo pošlje naprej v procesni element, ki z metodo podpornih vektorjev za vsako od naučenih kategorij preveri ali značilka pripada tej kategoriji. Zadnji procesni element obdelane rezultate zapakira v ustrezen odgovor originalnemu uporabniku.
Zgornja topologija je povsem zaporedne narave in je zato odzivni čas sistema vsota vseh procesnih elementov:

$$t_{odziv} = t_{LHOP} + t_{HoC} + t_{SVM} + \bar{t} \quad (1)$$

Vendar vsak od elementov lahko zaradi narave Storm-a teče v svojem procesu in tako omogoči delovanje sistema po principu cevovoda. Prav tako lahko za vsak procesni element skrbi več procesov, zaradi česar se lahko istočasno obdeluje več zahtev naenkrat.

6. ZMOGLJIVOST SISTEMA

Zmogljivost podsistema za porazdeljeno učenje smo preverili s postavitvijo Hadoop sistema na treh računalnikih, kjer ima vsak 35 vozlišč (40 CPU jeder) in 80 GB spomina. Za učno množico smo izbrali podatkovno zbirko slik Caltech-101 in naučili nekaj več kot 100 različnih kategorij. Za učenje 18248 slik (polovica od teh slik je prezrcaljenih) na 105 vozliščih smo potrebovali 37 minut, od tega je bilo 10 minut porabljenih za sam zagon, saj le-ta ni bil spisan povsem optimalno.

Tabela 1: Povprečen odzivni čas procesiranja zahteve po procesnih elementih.

t_{LHOP}	t_{HoC}	t_{SVM}	$\bar{t}$	t_{odziv}
1547 ms	85 ms	462 ms	48 ms	***2144 ms***

Podsistem tokovnega procesiranja v realnem času smo ovrednotili na enem računalniku z 8 procesnimi jedri in 16 GB spomina. Za topologijo smo namenili 4 različne procese, tako da je vsak procesni element potekal v svojem procesu. Rezultati testiranj so podani v tabeli 1. Podane vrednosti predstavljajo povprečje 20 ponovitev. V primeru implementacije na podobni konfiguraciji kot je bila uporabljena za sistem Hadoop (105 vozlišč), lahko po teoriji strežnega procesa [8] izračunamo, da je ob povprečnem času procesiranja 2144 ms maksimalno število zahtevkov, ki bi jih sistem še lahko sprejel in obdelal v tem času, 48 zahtev na sekundo.

7. ZAKLJUČEK

V članku smo predstavili arhitekturo, ki omogoča porazdeljeno procesiranje kategorizacije vizualnih objektov in služi za procesiranje zahtevkov iz spletne storitve. Implementirali smo kategorizacijo objektov z uporabo značilke HoC [5] in metode podpornih vektorjev [1]. Arhitekturo smo razdelili na dva dela: *porazdeljeno učenje* in *tokovno procesiranje v realnem času*. Prvi del smo implementirali v MapReduce [2] domeni s pomočjo implementacije Hadoop [4] in pokazali, da lahko celotno podatkovno zbirko Caltech-101 naučimo v manj kot 40 minutah na konfiguraciji računalnikov s 105 vozlišči. Za tokovno procesiranje v realnem času smo uporabili sistem Storm [6] in implementirali obdelavo spletnih zahtev kot Storm aplikacijo. Pokazali smo, da z našim sistemom dosežemo povprečen odzivni čas v rangu dveh sekund ter, da bi ob konfiguraciji računalnikov s 105 vozlišči lahko obdelali do 48 zahtev na sekundo.
V prihodnje želimo nadgraditi sistem z dodatnimi algoritmi računalniškega vida, specifično z iskanjem slik po vsebini, tako da bi upoštevali tudi informacijo o prepoznanih objektih na slikah.

LITERATURA

1. C.-C. Chang and C.-J. Lin. *Libsvm: A library for support vector machines.* ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Programska oprema na voljo na http://www.csie.ntu.edu.tw/ cjlin/libsvm.

2. J. Dean, S. Ghemawat, and G. Inc. *Mapreduce: simplified data processing on large clusters*. V OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation. USENIX Association, 2004.

3. F. F.-H. Nah. *A study on tolerable waiting time: how long are web users willing to wait?* Behaviour & IT, 23(3):153–163, 2004.

4. T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

5. D. Tabernik, M. Kristan, M. Boben, and A. Leonardis. *Learning statistically relevant edge structure improves low-level visual descriptors*. V International Conference on Pattern Recognition, 2012.

6. *http://storm-project.net*
   Informacije o sistemu Storm za porazdeljeno procesiranje v realnem času-

7. S. Fidler and A. Leonardis. *Towards scalable representations of object categories: Learning a hierarchy of parts.* V CVPR. IEEE Computer Society, 2007.

8. A. O. Allen. *Probability, statistics, and queueing theory with computer science applications.* Academic Press Professional, Inc., San Diego, CA, USA, 1990.