

SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5461

**Usporedba brzine pristupa velikoj  
količini podataka smještenih u  
lokalnom datotečnom sustavu i  
udaljenoj radnoj memoriji**

Ivan Đerek

Zagreb, rujan 2018.

**SVEUČILIŠTE U ZAGREBU**  
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**  
**ODBOR ZA ZAVRŠNI RAD MODULA**

Zagreb, 13. ožujka 2018.

## **ZAVRŠNI ZADATAK br. 5461**

Pristupnik: **Ivan Đerek (0036480761)**  
Studij: Računarstvo  
Modul: Programsко инженерство и информациони системи

Zadatak: **Usporedba brzine pristupa velikoj količini podataka smještenih u lokalnom datotečnom sustavu i udaljenoj radnoj memoriji**

Opis zadatka:

Proučiti i opisati dostupne usporedbe radnih svojstava računalnih programa kod pristupa istovrsnim podacima smještenim u različitim vrstama spremničkog prostora: datotečnom sustavu i radnoj memoriji računala. Proučiti i opisati načela rada sustava za pristup podacima smještenim na udaljenim računalima. Programski ostvariti radni okvir za usporedbu radnih svojstava računalnih programa tijekom pristupa velikoj količini podataka koji su u jednom slučaju smješteni u datotečnom sustavu lokalnog računala, a u drugom u radnoj memoriji skupine udaljenih računala. Ispitivanja provesti odvojeno za operaciju spremanja i operaciju dohvata podataka te za različite tehnologische i konfiguracijske parametre sustava: datotečni sustav ostvaren čvrstim diskom, tehnologijom SSD, odnosno flash memorijom; pristup podacima s osobnog računala i pametnog telefona pogonjenog operacijskim sustavom Android; te povezanost pristupnog i spremišnih računala lokalnom mrežom i javnom telekomunikacijskom mrežom. Opisati arhitekturu i programsko ostvarenje ispitnog sustava te dobivene rezultate ispitivanja. Navesti korištenu literaturu i primljenu pomoć.

Zadatak uručen pristupniku: 16. ožujka 2018.

Rok za predaju rada: 15. lipnja 2018.

Mentor:

\_\_\_\_\_  
Doc. dr. sc. Dejan Škvorc

Djelovodja:

\_\_\_\_\_  
Doc. dr. sc. Mirjana Domazet-Lošo

Predsjednik odbora za  
završni rad modula:

\_\_\_\_\_  
Izv. prof. dr. sc. Ivica Botički



# SADRŽAJ

<b>1. Uvod</b>	<b>1</b>
<b>2. Mediji za spremanje podataka</b>	<b>2</b>
2.1. Radna memorija . . . . .	2
2.2. Trajna memorija . . . . .	3
2.2.1. Čvrsti disk . . . . .	3
2.2.2. SSD memorija . . . . .	4
2.2.3. Cjenovna usporedba SSD memorije i čvrstog diska . . . . .	6
<b>3. Komunikacija računalnom mrežom</b>	<b>7</b>
3.1. Mrežna priključna točka . . . . .	8
3.2. Programiranje s mrežnim priključnim točkama . . . . .	9
<b>4. Programsко ostvarenje i rezultati sustava za ispitivanje</b>	<b>12</b>
4.1. Sustav sa strogim potvrđivanjem . . . . .	15
4.1.1. Rezultati . . . . .	16
4.2. Sustav bez potvrđivanja . . . . .	18
4.2.1. Rezultati . . . . .	18
4.3. Sustav s minimalnim potvrđivanjem . . . . .	22
4.3.1. Rezultati . . . . .	23
4.4. Korištenje sustava . . . . .	26
<b>5. Zaključak</b>	<b>28</b>
<b>Literatura</b>	<b>29</b>

# 1. Uvod

Radna memorija računala najbrža je memorija na računalu, ali je skupa i nije prigodna za trajno čuvanje podataka. Zbog toga računala imaju relativno ograničenu radnu memoriju u usporedbi s trajnom memorijom. Kod upravljanja većim količinama podataka može se dogoditi da radna memorija nije dovoljna pa je dio podataka potrebno držati na nekom drugom mediju, najčešće je to trajna memorija računala. Drugi pristup je koristiti radnu memoriju udaljenog računala i kako bi povećali ukupni kapacitet memorije. Cilj ovog rada je utvrditi mogu li se pristupanjem radnoj memoriji udaljenog računala dobiti bolji rezultati nego pristupanjem lokalnom datotečnom sustavu.

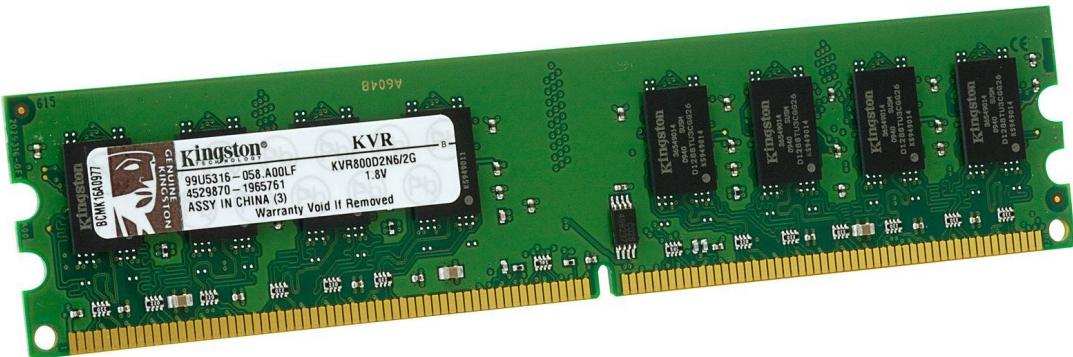
U sljedećem poglavlju ovog rada predstavljaju se različiti mediji za spremanje podataka, gdje su prikazane prednosti i mane pojedinih vrsta memorija. Nakon toga opisuje se metoda komunikacije između dvaju računala i prikazuje kako se pristupa programiranju kada je potreban sustav koji sadrži komunikaciju s udaljenim računalom. Nапослјетку je opisano programsko ostvarenje sustava za ispitivanje brzina pristupa različitim memorijama i udaljenom računalu gdje je utvrđeno kako je trajna memorija računala ipak u većini situacija brža, ali je, uz pravi pristup, moguće brže poslati podatke na udaljeno računalo u lokalnoj mreži, nego spremiti ih na računalo.

## 2. Mediji za spremanje podataka

U ovom poglavlju obrađene su temeljne karakteristike različitih medija za spremanje podataka. Detaljnije, opisane su radna memorija (eng. *Random access Memory*, skraćeno RAM), čvrsti disk (eng. *Hard Disk Drive*, skraćeno HDD) i SSD memorija (eng. *Solid-State Drive*), te usporedba u cijeni između trajnih memorija.

### 2.1. Radna memorija

Radna memorija koristi se u računalima kako bi se na nju učitali programi, operativni sustav i podaci koji se trenutno koriste, kako bi im se moglo brzo pristupiti[4]. Radna memorija je značajno brža od trajne memorije ali zadržava podatke samo dok ima pristup napajanju, što znači da nije efikasna niti praktična za spremanje podataka na duže vrijeme[3]. Radna memorija je toliko brza jer se bilo kojem podatku na memoriji može pristupiti direktno, neovisno o fizičkoj lokaciji na memoriji. Zato se uz poznatu adresu podatka na radnoj memoriji podatak može pročitati ili zapisati bez potrebe za prolaskom kroz ostale podatke, čime se značajno ubrzavaju operacija čitanja i pisanja. To je moguće zbog arhitekture radne memorije. Radna memorija posložena je u stupce i redove ćelija od kojih svaka ćelija sadrži jedan bit. Prilikom pristupanja podacima kontroler radne memorije prima adresu zapisa, nakon čega se određuju stupac i redak ćelije s koje se čita kako bi se podaci s pripadnih ćelija mogli zapisati na izlaz radne memorije. Najbitnija karakteristika radne memorije je ta da je puno brža od trajne memorije. Brzina radne memorije je bitna za rad sustava jer se u nju učitavaju podaci potrebni za izvršavanje procesa na računalu. Na slici 2.1 prikazan je primjer kako izgleda radna memorija.



Slika 2.1: Radna memorija

## 2.2. Trajna memorija

Pod trajnu memoriju računala spada memorija koja se koristi za trajno skladištenje podataka na računalu. Najčešće implementacije su čvrsti disk i ssd memorija. Trajna memorija je uvijek sporija od radne memorije, ali je zato uvijek značajno većeg kapaciteta. Ove razlike između trajne memorije i radne memorije postoje jer one nemaju istu namjenu u računalu. Dok se radna memorija koristi za programe i podatke koji su trenutno potrebni za izvođenje procesa, trajna memorija koristi se za skladištenje podataka između korištenja. Kada se podaci s lokalne memorije koriste, učitavaju se u radnu memoriju kako bi pristup istim podacima bio brži i spremniji za korištenje, u suprotnom bi procesor morao čekati da se učita podatak s lokalne memorije i time bi se značajno usporio rad računala. Trajnu memoriju zato koristimo ako želimo trajno spremiti podatak bez da ga izgubimo pri gašenju sustava. Problem nastaje ako se u radnu memoriju ne mogu učitati svi potrebni podaci, u tom je slučaju potrebno dio podataka privremeno zapisati na trajnu memoriju kako bi se oslobodio prostor za nove podatke. Budući da u toj situaciji dolazi do potrebe za češćim čitanjem i pisanjem na trajnu memoriju, brzina trajna memorije igra veliku ulogu. Ali nisu sve trajne memorije jednake. Čvrsti diskovi se koriste već duže vrijeme i imaju značajan kapacitet te relativno nisku cijenu. Novija ssd memorija u istom cjenovnom razredu ima značajno niži kapacitet, ali jednakoj tako i brži pristup podacima.

### 2.2.1. Čvrsti disk

Čvrsti disk[9] magnetni je disk podijeljen u male površine koje mogu individualno biti magnetizirane ili demagnetizirane. Na taj način se spremaju bitovi podataka koji ostaju zapisani čak i bez napajanja. Na slici 2.2 prikazan je primjer takvog diska. Čitanje i pisanje podataka s diska obavlja se pomoću magnetske glave koja se može pomicati

između centra i ruba čvrstog diska. Svim dijelovima diska može se pristupiti jer se disk konstantno okreće dok je aktivan. Jedan od razloga zašto je čitanje i pisanje s čvrstog diska sporije je taj što da bi se pristupilo podacima, glava se mora fizički pomaknuti te disk okrenuti na pravu lokaciju, vrijeme potrebno da se nađe podatak zove se vrijeme traženja. Vrijeme traženja ovisi o brzini vrtnje diska, vremenu da se magnetska glava pomakne na točnu lokaciju i načinu zapisivanja ili čitanja podataka[1]. Osim što takav mehanički pristup uzrokuje sporije čitanje i zapis podataka, on utječe i na pouzdanost ovog medija. Kako ima puno pokretnih dijelova može se dogoditi da se neki od dijelova mehanički ošteti i tada se više ne može pristupiti podacima na disku. Unatoč tim manama čvrsti diskovi su i dalje popularni oblik spremanja podataka, uglavnom zbog cijene i duge tradicije korištenja.



**Slika 2.2:** Primjer čvrstog diska

### 2.2.2. SSD memorija

SSD memorija koristi nili (eng. *NAND*)<sup>1</sup> logičke sklopove implementirane tranzistorima kako bi trajno spremila podatke. Ta vrsta memorije ponekad se još zove i brza

---

<sup>1</sup>(not and) logički sklop koji obavlja operaciju suprotnu od operacije i (eng. *AND*)

memorija (eng. *flash memory*) zbog toga što je brža u odnosu na čvrsti disk. Za razliku od radne memorije, tranzistori na SSD memoriji zadržavaju informaciju čak i kada nisu pod napajanjem, zbog čega se takva memorija može koristiti za lokalni datotečni sustav. Budući da ova memorija nema pokretne dijelove, podacima je moguće pristupiti puno brže nego na čvrstom disku jer im se, poput kod radne memorije, može pristupiti direktno. Dodatna prednost manjka pokretnih dijelova je ta što se time smanjuje mogućnost mehaničkog oštećenja, a time se dobiva i na stabilnosti memorije. Tranzistori u takvoj memoriji omogućuju zapis podataka pomoću različitih razina napona, koji se neće promijeniti kada se ukine dovod struje do diska[5]. Naravno, osim što je takva memorija sporija od radne memorije, i dalje se podaci moraju najprije učitati na radnu memoriju kako bi ih koristili. Danas SSD memorija postaje sve popularnija i već je standardno da računala imaju SSD memoriju barem kao glavnu memoriju, time osiguravaju brže učitavanje operativnog sustava računala i brži rad aplikacija koje se spremaju na glavnu memoriju.



**Slika 2.3:** Primjer SSD memorije

### 2.2.3. Cjenovna usporedba SSD memorije i čvrstog diska

Iako je ustanovljeno da je SSD memorija puno brža i pouzdanija, čvrsti diskovi su i dalje prisutni na tržištu. Glavni razlog tome je cijena. Danas je popularno imati SSD kao glavnu memoriju i uz njega sporedni čvrsti disk visokog kapaciteta. Time se postižu optimalne performanse sustava i istovremeno zadovoljavaju sve veće potrebe za velikim kapacitetom memorije. Najbolji način za usporedbu cijena je usporedba cijene po gigabajtu na memoriji. Tablica 2.4 prikazuje cijenu nekih čvrstih diskova u randima<sup>2</sup>. Može se vidjeti kako je kod čvrstih diskova cijena po gigabajtu najniža pri 4TB, dok je kod SSD memorije pri 1TB i tu je razlika u cijeni po gigabajtu skoro 10 puta veća[7], dok se za cijenu čvrstog diska od 8TB može kupiti SSD memorija od najviše 1TB. Iz toga je lako zaključiti zašto se čvrsti diskovi i dalje koriste u tako velikom broju.

Ime proizvoda	Kapacitet	Cijena	Rand po GB
<b>Čvrsti diskovi</b>			
Seagate Baracuda	500GB	R672	<b>R1.34</b>
Western Digital Blue	1TB	R629	<b>R0.63</b>
Seagate Barracuda	2TB	R929	<b>R0.46</b>
Seagate Barracuda	3TB	R1279	<b>R0.43</b>
Seagate Barracuda	4TB	R1599	<b>R0.40</b>
Seagate	5TB	R2474	<b>R0.49</b>
Western Digital Blue	6TB	R2637	<b>R0.44</b>
Seagate Ironwolf	8TB	#3865	<b>R0.48</b>
<b>SSD memorija</b>			
Western Digital Green	120GB	R649	<b>R5.41</b>
Crucial BX200	240GB	R1,199	<b>R5.00</b>
Kingston	480GB	R2,476	<b>R5.16</b>
Crucial MX500	500GB	R2,099	<b>R4.20</b>
Crucial MX300	1TB	R3,899	<b>R3.90</b>
Crucial MX500	2TB	R8,273	<b>R4.14</b>
Samsung 860 Evo	4TB	R20,951	<b>R5.24</b>

**Slika 2.4:** Tablica s primjerima cijena memorija

<sup>2</sup>Službena valuta Južnoafričke Republike

### 3. Komunikacija računalnom mrežom

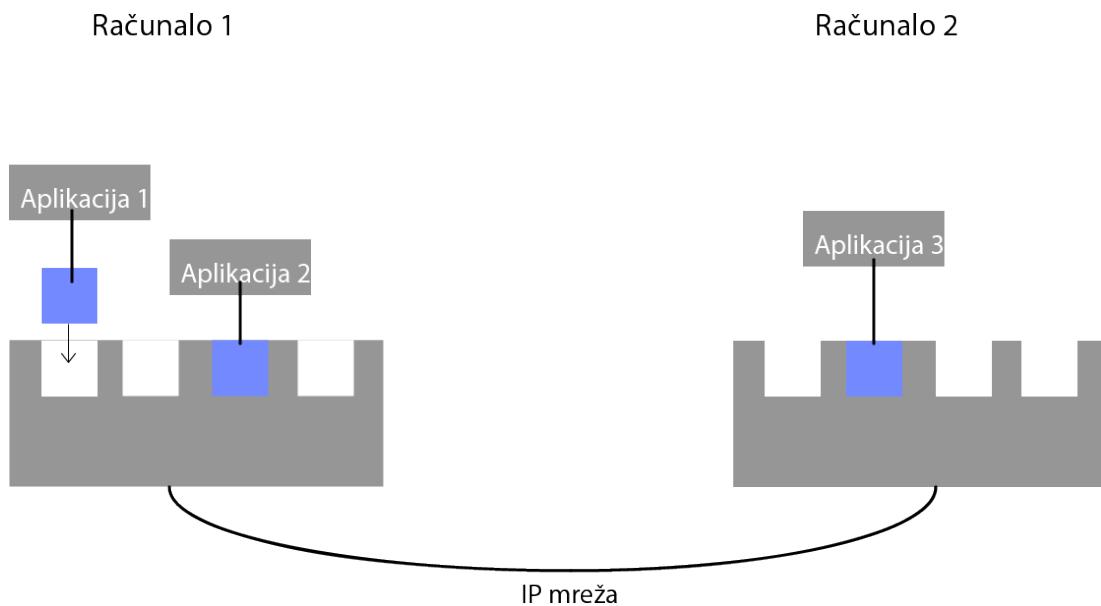
U moderno vrijeme sve su popularnija različita rješenja poput spremanja podataka u "oblaku" (eng. *Cloud storage*)<sup>1</sup> i korištenja resursa s udaljenih računala. Razlog tome je što je prijenos podataka putem javne mreže sve brži i praktičniji za korištenje. Jednom kada su podaci na udaljenom računalu, mogu se spremiti na bilo koji od oblika trajne memorije ili na radnu memoriju tog računala. Najbrže se može pristupiti podacima koji se nalaze na radnoj memoriji udaljenog računala, odnosno oni podaci koji su već spremni za prijenos putem mreže. Ako je moguće pouzdano pristupati tim podacima onda je izmjena podataka podacima slična kao s trajnom memorijom. Naime kako bi se podaci koristili moraju se nalaziti na radnoj memoriji računala. Ako je tih podataka previše za radnu memoriju računala mora ih se spremiti na neku drugu memoriju. Uobičajeno rješenje je spremiti ih na lokalnu trajnu memoriju, ali ako se ti podaci nalaze u radnoj memoriji drugog računala postoje određene prednosti. Kada bi prijenos podataka preko mreže bio brži nego učitavanje s trajne memorije onda bi prednost bila očita, moguće bi bilo koristiti radnu memoriju računala na mreži koja se trenutno ne koristi i upotrijebiti je kao brzu memoriju za računalo na kojem je potrebna. Čak i ako je prijenos podataka sporiji činjenica je da bi se na udaljenom računalu ti podaci već nalazili u radnoj memoriji i moguće je obaviti dio posla paralelno na tom udaljenom računalu, time bi se ubrzao ukupan posao tako da se na kraju dohvati samo rezultat operacija. Zadnja velika prednost ovakvog rješenja je ta što, ako se omogući pristup podacima s udaljenog računala, njima se može pristupiti s više računala i lokacija.

---

<sup>1</sup>Popularan naziv za skladištenje podataka na mreži kako bi im se moglo pristupiti s raznih lokacija.

### 3.1. Mrežna priključna točka

Mrežna priključna točka (eng. *Network socket*) programsko je sučelje pruženo TCP (eng. *Transmission Control Protocol*)<sup>2</sup> ili UDP (eng. *User Datagram Protocol*) protokolom[6]. To je priključna točka za komunikaciju s udaljenim računalom definirana IP (eng. *Internet protocol*) adresom i pristupnom točkom (eng. *port*)<sup>3</sup>. Razlog iz kojeg je potrebna pristupna točka je taj što mrežna priključna točka ne predstavlja cijelo računalo nego samo proces koji je otvoren za komunikaciju. IP adresa određuje samo uređaj, a na njemu se može obavijati više procesa, kako bi se moglo spojiti na ispravan proces potrebna nam je pristupna točka[8]. Ako je bitno



**Slika 3.1:** Mrežne priključne točke omogućuju komunikaciju između aplikacija na udaljenim računalima

da podaci ostanu u prvobitnom stanju koristi se povezanost pomoću TCP/IP servisa. Naime iako je bitna brzina prijenosa podataka, ono što je bitnije je da se ti podaci cjelovito prenesu, inače mogu izgubiti svoju korisnost. Kod programiranja s priključnim točkama jedan proces je u ulozi poslužitelja, dok je drugi klijent, glavna razlika među njima je pri uspostavljanju veze, naime klijent mora znati IP adresu i pristupnu točku poslužitelja kako bi se spojio, nakon što je veza uspostavljena komunikacija je obostrana i poslužitelj i klijent su ravnopravni. Mrežna priključna točka može obavljati sljedeće 4 osnovne operacije: spajanje s udaljenom priključnom točkom, slanje podataka, primanje podataka i zatvaranje veze. Kod takve komunikacije potrebno je

<sup>2</sup>internet protokol kod kojeg se pazi da je poruka primljena

<sup>3</sup>identifikacija procesa ili mrežne usluge na računalu

prvo spojiti dva računala, što se započinje otvaranjem veze na poslužitelju. Kako bi se klijent spojio na poslužitelj, prvo mora znati IP adresu računala na kojem je poslužitelj i pristupnu točku procesa koji je u ulozi poslužitelja. Veza se zatim inicira sa strane klijenta koji se pokušava spojiti na poznatu IP adresu i pristupnu točku. Jednom kad je veza potvrđena može se započeti slanje podataka. Podaci se šalju putem izlaznih i ulaznih sljedova podataka (eng. *stream*) mrežnih priključnih točaka.

## 3.2. Programiranje s mrežnim priključnim točkama

Moguće je koristiti više različitih implementacija na bazi mrežnih priključnih točaka, ali kod svih je jednakto da sadrže osnovne operacije. U Java programskom jeziku koriste se `java.net.Socket` i `java.net.ServerSocket` klase[2]. Najbitnije metode `ServerSocket` klase:

1. `accept()`, vraća `Socket`, osluškuje vezu na mrežnoj priključnoj točki i prihvata ju
2. `bind(SocketAddress endpoint)`, vraća `void`, veže mrežnu priključnu točku na zadatu adresu koja se sastoji od IP adrese i pristupne točke
3. `close()`, vraća `void`, zatvara mrežnu priključnu točku

`ServerSocket` na sebe veže IP adresu i pristupnu točku koja mu se zada te počinje osluškivati vezu metodom `accept()`. Jednom kada `ServerSocket` primi zahtjev za spajanjem, prihvata ga i time dobiva referencu na mrežnu priključnu točku kako bi se mogli otvoriti sljedovi podataka(eng. *data stream*) između priključnih točaka. Najbitnije metode `Socket` klase su:

1. `connect(SocketAddress endpoint)`, vraća `void`, veže mrežnu priključnu točku na poslužitelja koji se nalazi na zadanoj IP adresi i pristupnoj točki
2. `bind(SocketAddress endpoint)`, vraća `void`, veže mrežnu priključnu točku na zadatu adresu koja se sastoji od IP adrese i pristupne točke
3. `getInputStream()`, vraća `InputStream`, `InputStream` nam je potreban za čitanje podataka koje prima mrežna pristupna točka
4. `getOutputStream()`, vraća `OutputStream`, `OutputStream` nam je potreban za slanje podataka na povezanu mrežnu pristupnu točku
5. `close()`, vraća `void`, zatvara mrežnu priključnu točku

Na klijentskoj aplikaciji potrebno je znati IP adresu i pristupnu točku na koji je vezan ServerSocket kako bi se iniciralo spajanje. Jednom kada je veza uspostavljena, klijent i poslužitelj mogu komunicirati. Komunikacija se obavlja putem sljedova podataka na priključnim točkama. Svaka strana ima svoj ulazni i izlazni slijed podataka koji je spojen na priključnu točku drugog procesa. Dalje se programira upravljanjem sljedova podataka. Primjer jednostavnih aplikacija poslužitelja i klijenta može se vidjeti na slikama 3.2 i 3.3.

```
Scanner scanner = new Scanner(System.in);

System.out.print("Enter port number: ");
int portNumber = scanner.nextInt();

int length = 1048576;
int bufferSize = 102400;

try (ServerSocket serverSocket = new ServerSocket(portNumber))
{
    System.out.println("Waiting for connection at: " + InetAddress.getLocalHost().getHostAddress());
    Socket clientSocket = serverSocket.accept();
    InputStream in = clientSocket.getInputStream();

    System.out.println("Client connected on port " + portNumber + ". Servicing requests.");
    for(int i = 0; i<1000; i++) {

        byte[] dataBytes = new byte[length];
        byte[] buffer = new byte[bufferSize];
        ByteBuffer dataBuffer = ByteBuffer.allocate(length + length);
        long n = length;
        int dataRead;
        while ((dataRead = in.read(buffer)) != -1) {
            n -= dataRead;
            dataBuffer.put(buffer, offset: 0, dataRead);
            if (n <= 0)
                break;
        }
        dataBytes = Arrays.copyOfRange(dataBuffer.array(), from: 0,length);
    }
} catch (IOException e) {
    System.out.println("Exception caught when trying to listen on port "
        + portNumber + " or listening for a connection");
    e.printStackTrace();
}
```

**Slika 3.2:** Isječak jednostavne poslužiteljske aplikacije

```

int length = 1048576;
int bufferSize = 102400;

Scanner scanner = new Scanner(System.in);
System.out.print("Enter host name: ");
String hostName = scanner.nextLine();
System.out.print("Enter port number: ");
int portNumber = scanner.nextInt();
scanner.nextLine();

byte[] dataBytes = new byte[length];
new Random().nextBytes(dataBytes);

try (Socket serverSocket = new Socket(hostName, portNumber)) {
    System.out.println("Connected to: " + serverSocket.toString());
    byte[] buffer = new byte[bufferSize];
    OutputStream out = serverSocket.getOutputStream();

    long time = System.currentTimeMillis();
    for (int i = 0; i<1000; i++) {
        ByteArrayInputStream byteStream = new ByteArrayInputStream(dataBytes);
        int count;
        while ((count = byteStream.read(buffer)) > 0 && serverSocket.isConnected()) {
            out.write(buffer, off: 0, count);
        }
    }
    time = (System.currentTimeMillis() - time)/1000;
    System.out.println("average time for " + String.valueOf(length) + "kB :" + String.valueOf(time) + "ms.");
} catch (IOException e) {
    System.out.println("Exception caught when trying to listen on port "
        + portNumber + " or listening for a connection");
    e.printStackTrace();
}
}

```

**Slika 3.3:** Isječak jednostavne klijentske aplikacije

## **4. Programsко ostvarenje i rezultati sustava za ispitivanje**

Za potrebe istraživanja za ovaj rad bilo je potrebno napraviti jednostavnu aplikaciju koja ima pristup radnoj memoriji na udaljenom računalu. Aplikacija mora spremati i čitati podatke s udaljenog računala i lokalne memorije te zabilježiti vremena pisanja i čitanja podataka kako bi se moglo usporediti razlike u brzini pristupa podacima na različitim memorijama. Aplikacija je rađena u Java programskom jeziku koristeći IntelliJ Idea razvojno okruženje za računalne aplikacije i C# programskom jeziku u sustavu Xamarin koristeći Visual Studio razvojno okruženje za Android aplikaciju.

Osnovna ideja sustava koji testira brzine prijenosa podataka je da se zabilježi vrijeme prije prijenosa podataka, obavi prijenos i onda odredi koliko je vremena prošlo. Budući da prijenos ne traje svaki put jednako, za precizniju informaciju potrebno je izračunati prosjek vremena trajanja iz puno ponovljenih prijenosa s istim podacima. Radi toga se u klijentu koji radi testiranje, prijenos podataka ponavlja i na kraju se ukupno vrijeme dijeli s brojem prijenosa, kao kod primjera pisanja podataka na poslužitelj koji je prikazan na slici 4.1. U sklopu istraživanja provodi se više različitih testova. Na slici 4.2 prikazani su odnosi klijentskih aplikacija s poslužiteljem. Za svaku prikazanu liniju mogu se obaviti operacije pisanja i čitanja.

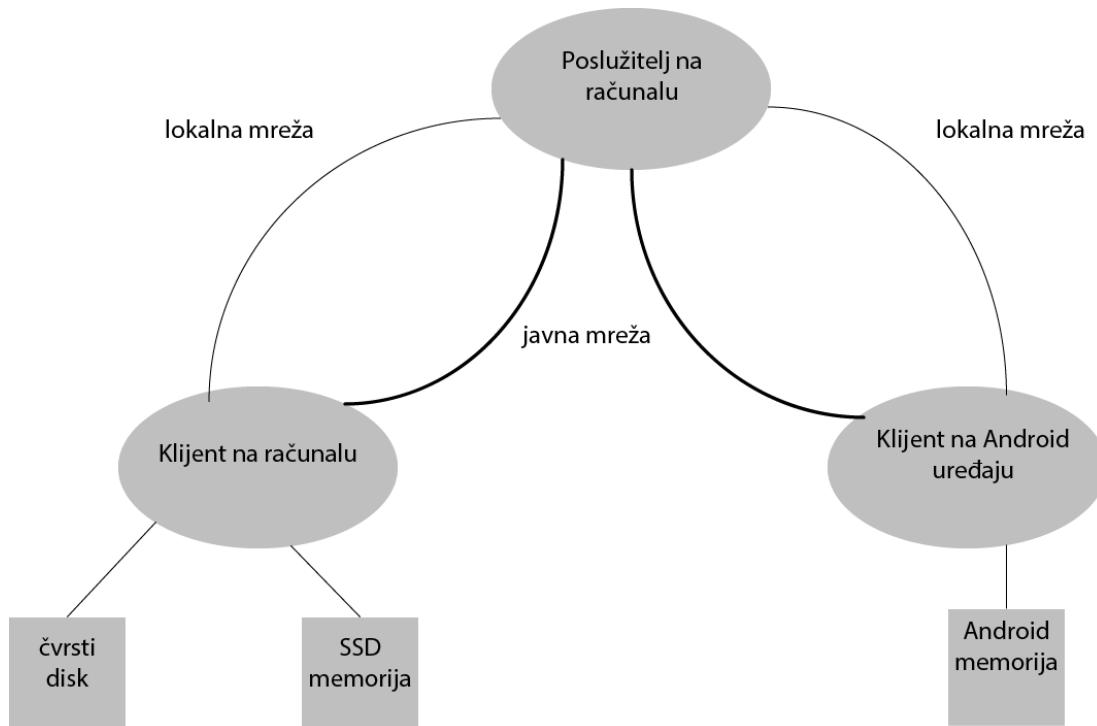
Aplikacija za računalo napisana u Java programskom jeziku sastoji se od tri klase: Server, AutoClient i ClientHelper. Klase koje se mogu pokrenuti su Server i AutoClient dok ClientHelper opisuje metode korištenja podataka na klijentu. AutoClient i Client se razlikuju u tome što je za AutoClient potrebno unijeti samo osnovne podatke i dalje generira veliki broj testova, dok Client traži korisnika da stalno unosi naredbe. Aplikacije su napisane kao jednostavne aplikacije za terminal. Server ima dva objekta koji predstavljaju mrežne priključne točke, jedan predstavlja priključnu točku na poslužitelju, dok drugi predstavlja priključnu točku na klijentu. Prvo je potrebno stvoriti mrežnu priključnu točku na poslužitelju i dodijeliti mu pristupnu točku, nakon čega se čeka da klijent započne spajanje. Kada se klijent spoji poslužitelj dobiva informacije

```

System.out.println("Receiving from server");
ClientHelper.FileTestData testData = null;
averageTime = System.currentTimeMillis();
for (int x = 0; x < repeatNumber; x++) {
    if(!useHeaders)
        testData = tester.reachFileFromServer(serverSocket, fileName);
    else
        testData = tester.reachFileFromServer2(serverSocket, fileName);
    dataBytes = testData.data;
}
averageTime = (System.currentTimeMillis() - averageTime)/repeatNumber;
tester.saveResult(fileSize, averageTime,testName, hostName.toLowerCase()
.equals("localhost")?"local net":"net", action: "server read");

```

**Slika 4.1:** Isječak koda za testiranje brzine slanja podataka na poslužitelj koristeći veći broj uzastopnih testova



**Slika 4.2:** Prikaz odnosa među aplikacijama i njihovih pripadnih memorija

o klijentu kako bi mogao koristiti ulazne i izlazne sljedove podataka povezane s klijentom. Dalje poslužitelj prima informaciju o načinu na koji će razmjenjivati podatke (putem više poruka ili pomoću zaglavlja koja nose više informacija) i zatim ponavlja petlju u kojoj od klijenta prima poruku s informacijom o naredbi koju radnju klijent želi obaviti. Svaki klijent ima samo referencu na mrežnu pristupnu točku od poslužitelja. Metode za pisanje i čitanje s trajne memorije mogu se vidjeti na slici 4.3.

Aplikacija za android uređaje napisana u C# programskom jeziku sastoji se od jednog zaslona i pomoćne klase koja upravlja komunikacijom s poslužiteljem. Android

```

public void saveFileToMemory(byte[] dataBytes, String fileName, String path) throws IOException{
    Path file = Paths.get( first: path+"/"+fileName);
    try{
        Files.write(file, dataBytes);
        return;
    }catch (IOException e){
        System.out.println(e.toString());
    }catch ( NullPointerException a){
        System.out.println("data not initialised!");
    }
    return ;
}

public FileTestData reachFromMemory(String fileName, String path){

    Path nFile = Paths.get( first: path+"/"+fileName);
    try{
        byte[] dataBytes = Files.readAllBytes(nFile);
        return new FileTestData(fileName, dataBytes);
    }catch (IOException e){
        System.out.println(e.toString());
        e.printStackTrace();
    }catch ( NullPointerException a){
        System.out.println("data not initialised!");
        a.printStackTrace();
    }
    return null;
}

```

**Slika 4.3:** Isječak koda za pisanje i čitanje podataka s trajne memorije računala

aplikacija ne može se koristiti kao poslužitelj, nego se ponaša jednako kao AutoClient u aplikaciji na računalu. Aplikacija sadržava ClientHelper klasu koja je jednaka kao ona na računalu, samo prepisana u C#, zato se klijent na mobilnom uređaju ponaša jednako kao onaj na računalu u komunikaciji s poslužiteljem. Jedina razlika u ClientHelper klasi na android aplikaciji je u pristupu trajnoj memoriji zbog specifičnosti android operativnog sustava, primjer se može vidjeti na slici 4.4. Sučelje aplikacije sastoji se od polja za upis IP adrese računala, polja za upis pristupne točke, gumba za pokretanje testova i prikaza na kojem se vide informacije o testiranju u tijeku.

```

public static void saveFileToMemory(byte[] dataBytes, String fileName){
    var startTime = DateTime.Now;
    var documents = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
    var filename = Path.Combine(documents, fileName);
    File.WriteAllBytes(filename, dataBytes);
    return;
}

public static FileTestData reachFromMemory(String fileName){
    var startTime = DateTime.Now;
    var documents = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
    var filename = Path.Combine(documents, fileName);
    byte[] dataBytes = File.ReadAllBytes(filename);
    return new FileTestData(fileName, dataBytes);
}

```

**Slika 4.4:** Isječak koda za pisanje i čitanje podataka s trajne memorije Android uređaja

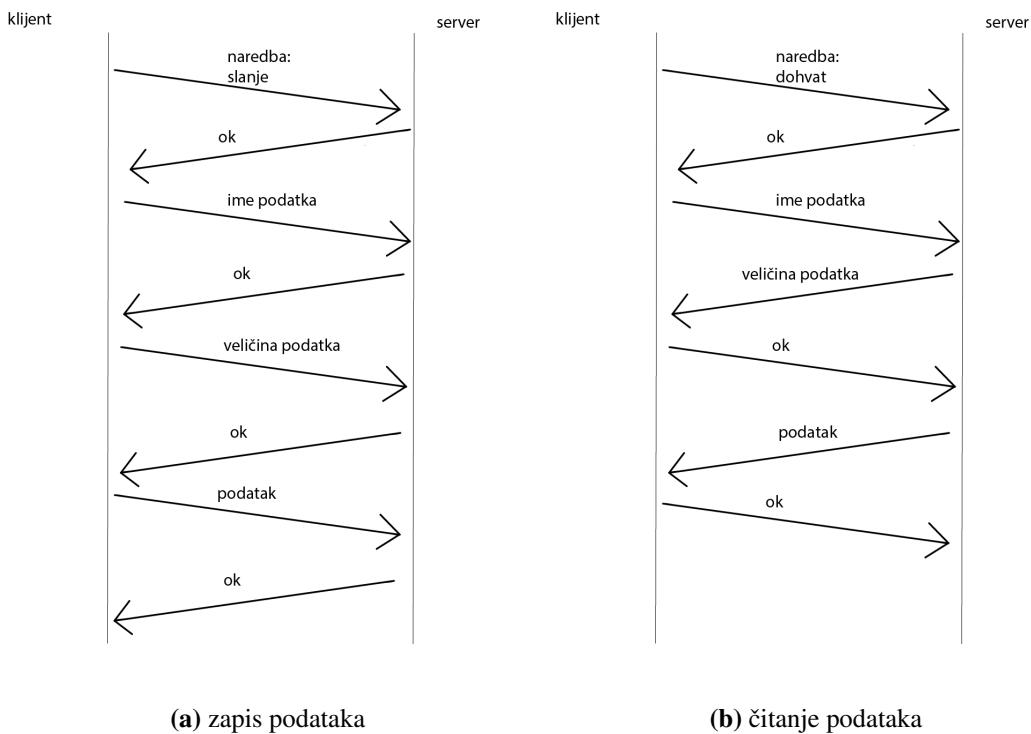
## 4.1. Sustav sa strogim potvrđivanjem

Kako bi poslužitelj znao koju radnju treba obaviti moramo mu najprije poslati poruku sa sljedećom radnjom. Prvotna ideja je da se svaka poruka šalje zasebno te zatim čekaju potvrde od poslužitelja da je informacija primljena kako bi se nastavilo s radom. Protok komunikacije može se vidjeti na slici 4.5. Nakon slanja informacija potrebnih za čitanje podatka, podaci se šalju u dijelovima kako se ne bi preopteretio slijed podataka. Poslužitelj primljene podatke sprema u listu te koristi odvojenu listu naziva podataka s odgovarajućim indeksima. Ako poslužitelj primi podatak naziva koji već ima, prebrisat će se stari podatak. Veličina podataka koja se odjednom šalje je najviše pola MB<sup>1</sup>, naime ako se šalju pre veliki podaci od jednom, sustav često zapne u komunikaciji. Za svaku radnju na kraju se zapisuje rezultat koji sadrži:

1. vrijeme izvršavanja testa
2. vrstu testa (slanje ili primanje podataka na poslužitelj ili lokalnu memoriju)
3. veličinu podataka u KiB
4. proteklo vrijeme između početka i kraja zapisivanja ili čitanja podataka u ms
5. medij na kojem se zapisao ili nalazio podatak (mreža, čvrsti disk ili ssd memorija)
6. ime testa

---

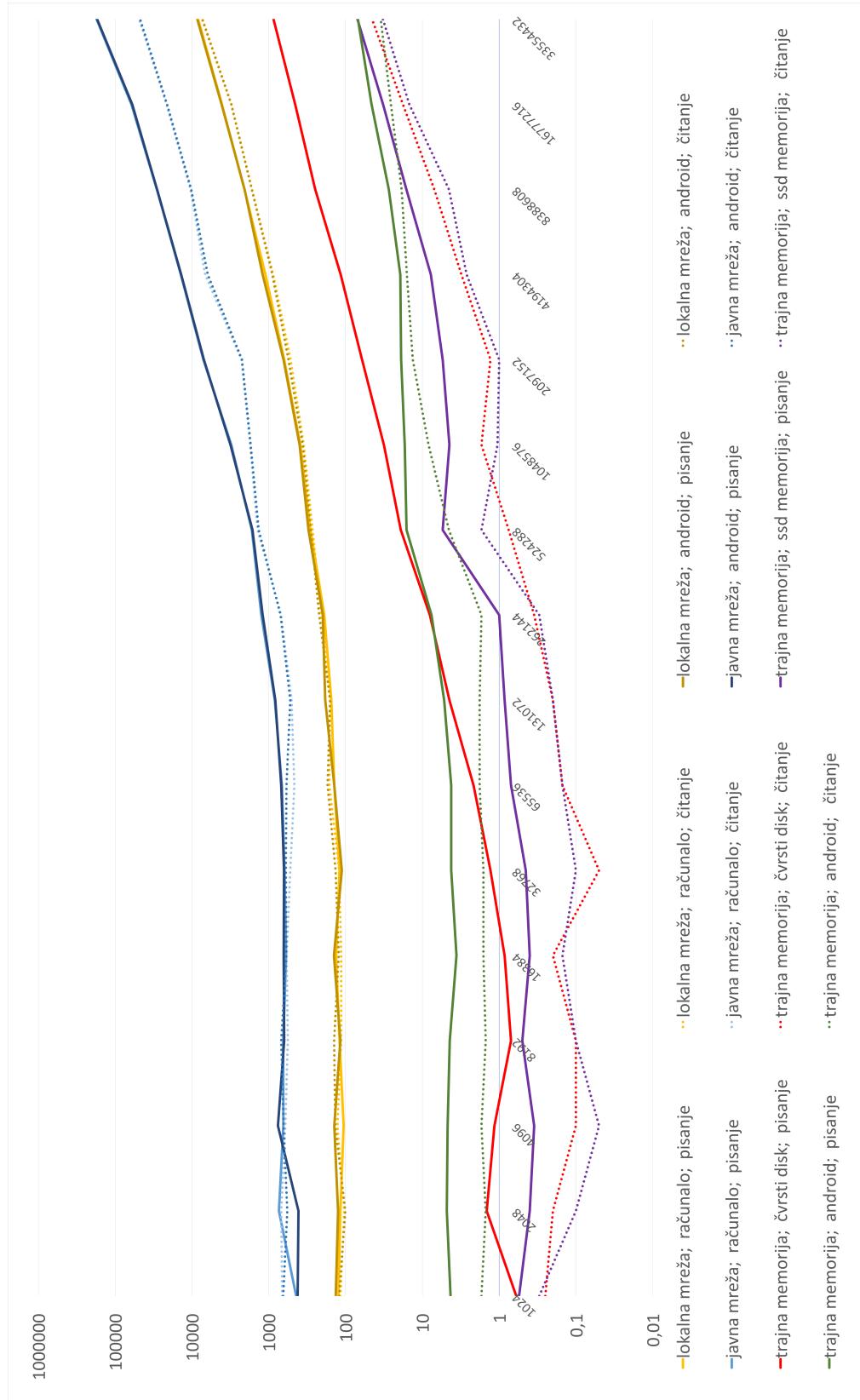
<sup>1</sup>mebi bajt



**Slika 4.5:** Grafički prikaz protokola za čitanje i pisanje podataka na udaljeno računalo

#### 4.1.1. Rezultati

Iz rezultata je vidljivo da ova implementacija pokazuje slabe rezultate pri razmjeni podataka s udaljenim računalom. Očekivano je da će razmjena podataka s računalom na udaljenoj mreži biti sporija, ali čak i na lokalnoj mreži, vremena zapisa i čitanja s udaljenog računala pokazala su se dužima od razmjene podataka s trajnom memorijom. Također iz grafa je vidljivo kako brzina razmjene podataka kod manjih podataka ne raste monotono u početku. Ono što se iz toga može zaključiti je da je protokol oko izmjene podataka usporio postupak pri manjim podacima. Zanimljivo je da nema velike razlike u brzini prijenosa podataka s poslužiteljem neovisno o tome da li se klijent nalazi na računalu ili android uređaju. Također je bitno primjetiti da kod android uređaja slabije raste vrijeme komunikacije s lokalnom memorijom nego kod ostalih datotečnih sustava, ali je zato kod manjih podataka sporiji od čvrstog diska i ssd memorije.



**Slika 4.6:** Brzine pisanja i čitanja svih stavki u sustavu sa strogim potvrđivanjem

## 4.2. Sustav bez potvrđivanja

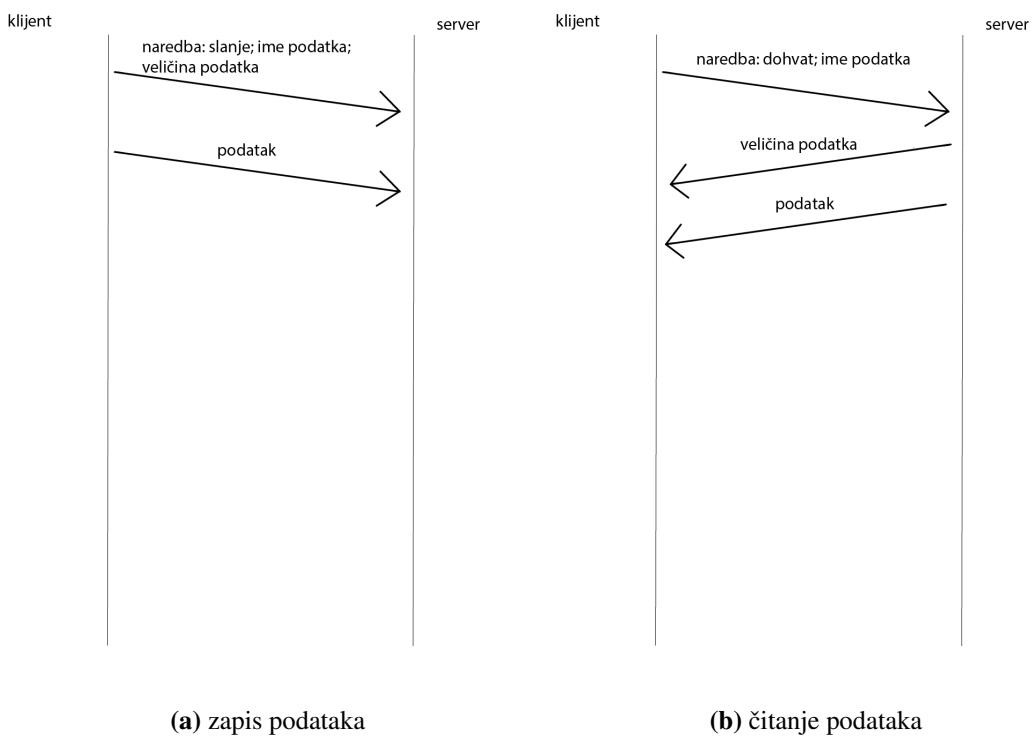
Kako bi se smanjio utjecaj protokola na brzinu prijenosa podataka minimizirana je komunikacija s poslužiteljem prije i poslije izmjene podataka. Podaci koji su nužni kako bi se mogao zapisati ili pročitati podatak su:

1. radnja koja se treba obaviti
2. naziv podatka
3. veličina podatka

Kako se ne bi bilo potrebe za stalnom komunikacijom dodane su dvije klase, `SocketMessageHelper` i `Header`. Klasa `Header` predstavlja zaglavlje koje se sastoji od navedenih podataka. `Header` ima metodu koja ga pretvara u polje bajtova od 55 mesta i ima konstruktor koji ga stvara iz polja bajtova. `SocketMessageHelper` služi za standardiziranje informacija o operacijama. U ovom sustavu je za zapis podataka na poslužitelj prije slanja samih podataka potrebno poslati samo zaglavlje nakon čega odmah počinje slanje podataka (Slika 4.7). Kako bi se minimiziralo vrijeme komunikacije, ni u jednom trenutku se od poslužitelja ne traži potvrda da je slanje gotovo. Za čitanje podataka od poslužitelja potrebno je dobiti informaciju o veličini podatka prije nego što se može započeti učitavati podatke, zbog čega čitanje u ovom slučaju ima jednu radnju više. Nema promjena u komunikaciji s lokalnim datotečnim sustavom jer se već koriste jednostavne ugrađene operacije. Jedini problem koji je nastao je taj da je kod nekih testiranja s podacima većim od 8MB došlo do pucanja veze zbog greške na slijedu podataka, ali u većini testiranja se to nije dogodilo i zato su se uspješno skupili podaci o brzini slanja podataka.

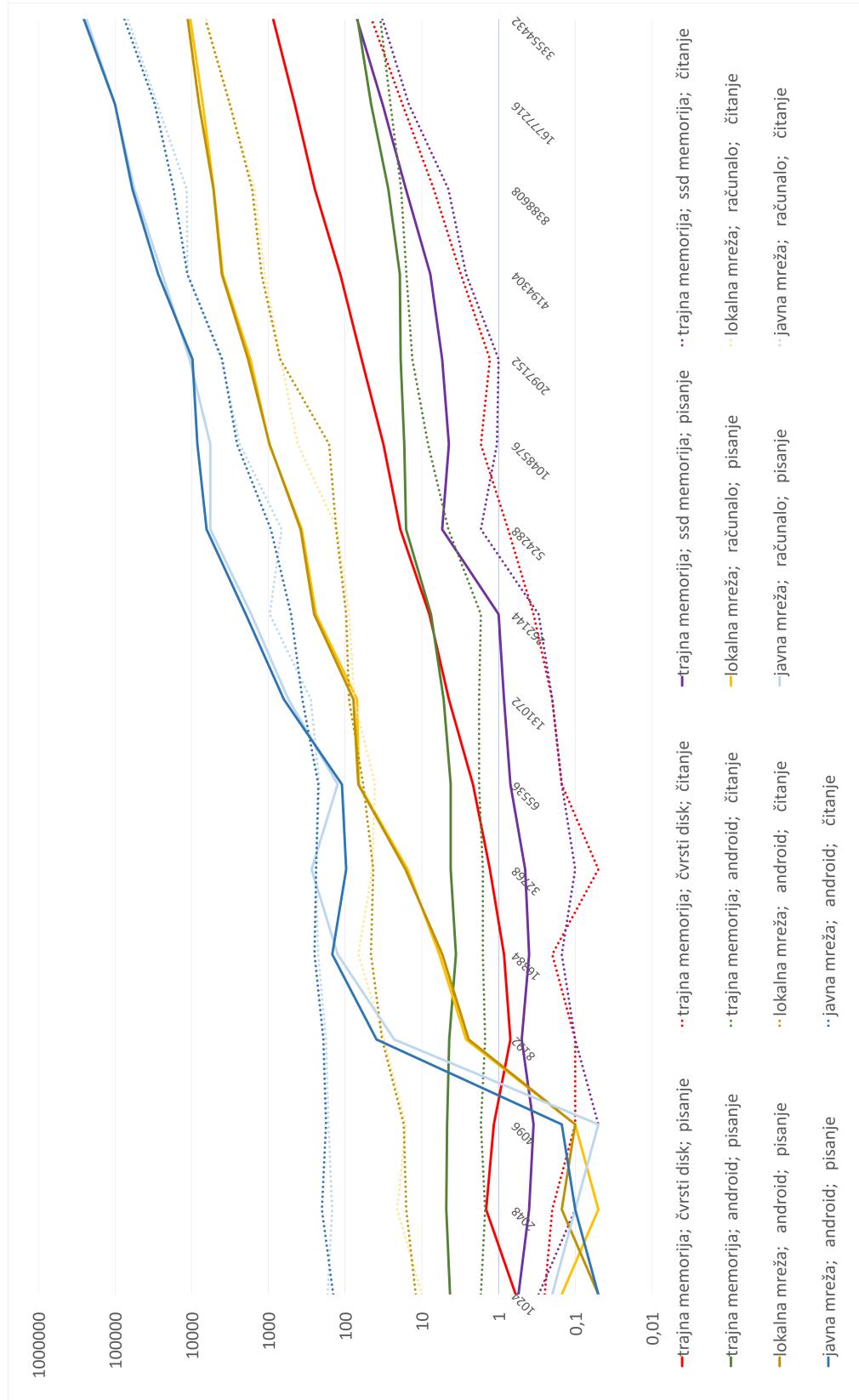
### 4.2.1. Rezultati

Ovom optimizacijom dobiveno je veliko smanjenje vremena potrebnog za izmjenu podataka s poslužiteljem kod manjih podataka. Ono što je posebno interesantno je da se kod podataka manjih od 8kB pokazalo kako je zapisivanje podataka na udaljeno računalo brže od zapisivanja na čvrsti disk ili memoriju android uređaja, čak i kod javne mreže. Naravno ovdje se klijent ne brine o tome da li je poslužitelj učitao cijeli podatak prije nego krene sa sljedećom radnjom i ne znamo je li podatak uspješno spremlijen na poslužitelju. Takvih problema nema s čitanjem jer klijent zna da li je pročitao cijeli podatak. Problem koji odmah možemo uočiti je to da kod zapisa većih podataka vrijeme raste u odnosu na prethodni test. To se događa jer je ovaj sustav



**Slika 4.7:** Grafički prikaz protokola za čitanje i pisanje podataka na udaljeno računalo u sustavu bez potvrđivanja

manje stabilan i opterećujemo izlazni slijed podataka bez da provjerimo je li poslužitelj sve uspio pročitati.



**Slika 4.8:** Brzine zapisa i čitanja svih stavki u sustavu bez potvrđivanja



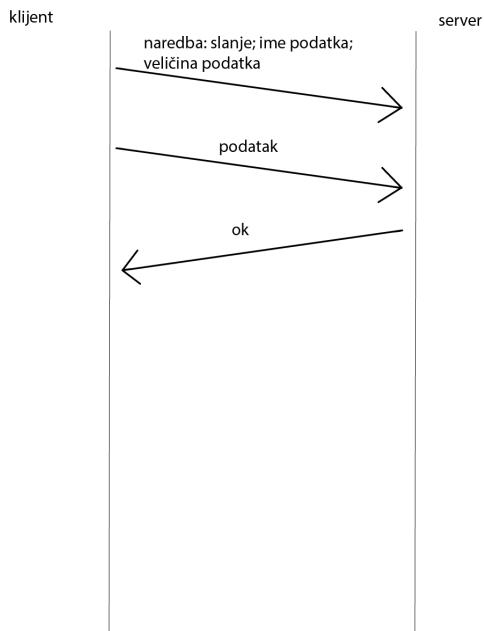
**Slika 4.9:** Usپoredба brzina sustava sa strogim potvrђivanjem i sustava bez potvrђivanja

### **4.3. Sustav s minimalnim potvrđivanjem**

Kako bi se sa sigurnošću moglo reći da je podatak koji je potrebno spremiti na poslužitelj stvarno bio uspješno poslan, potrebna je povratna informacija od poslužitelja (Slika 4.10). Naravno svaka dodatna izmjena podataka s poslužiteljem usporava ukupni postupak zapisivanja podataka. Jedna dodatna pozitivna stvar kod traženja povratne informacije na kraju slanja podataka na poslužitelj je ta što je osigurano da se neće slati novi podaci poslužitelju dok još nije pročitao posljednji podatak čime je dignuta stabilnost sustava. Iz grafa na slici 4.9 vidljivo je da je čitanje podataka s poslužitelja također pokazalo poboljšanje u zadnjem testu, a u toj operaciji postoji jedna operacija više<sup>2</sup>, zato se može zaključiti da je prihvatljivo da klijent na kraju svakog zapisa od poslužitelja traži potvrdu da je prijenos gotov.

---

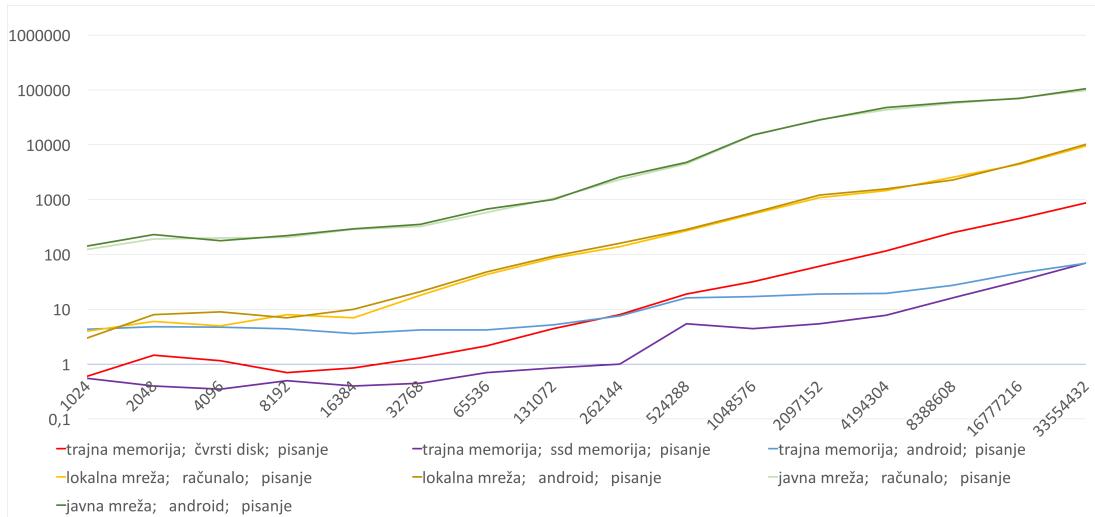
<sup>2</sup>primanje veličine podatka od poslužitelja



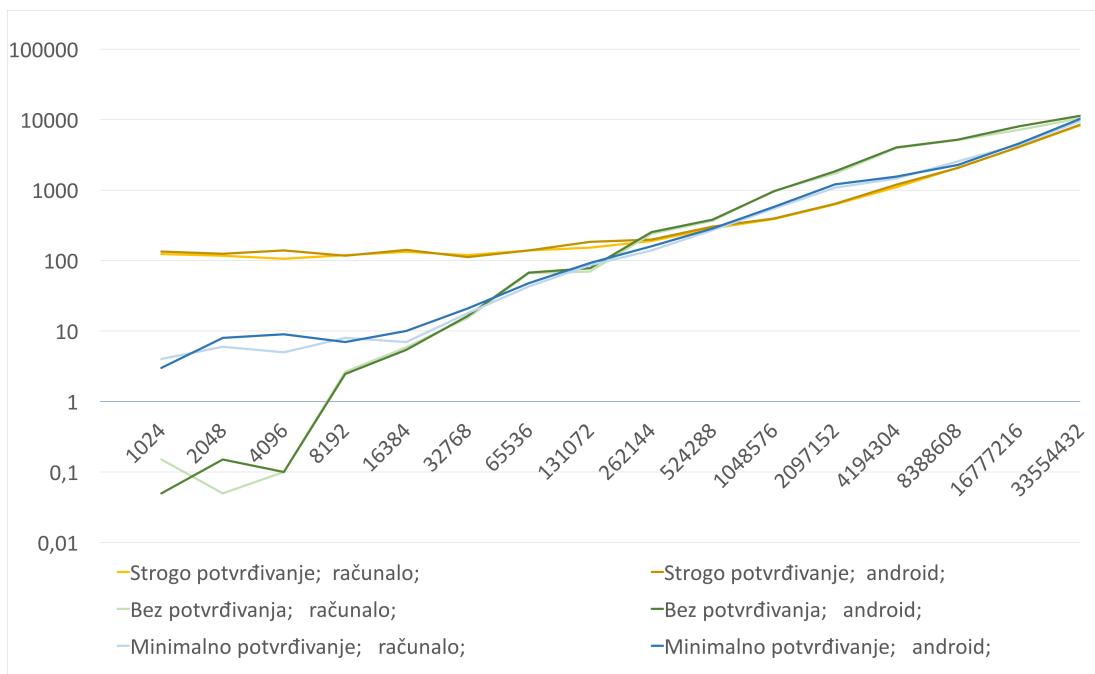
**Slika 4.10:** Grafički prikaz protokola s minimalnim potvrđivanjem

### 4.3.1. Rezultati

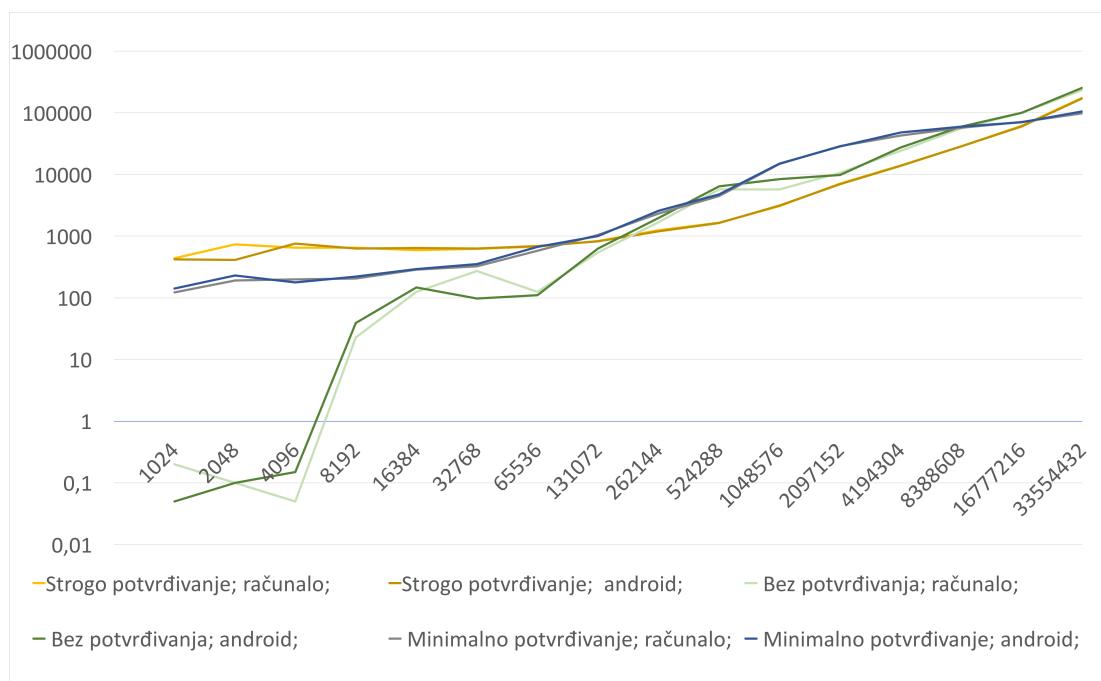
Kako je promjena bila samo u zapisivanju na poslužitelja, ne prikazuju se rezultati čitanja za ovaj test. Vidljiv je rast u vremenu potrebnom da se završi postupak slanja manjih podataka u odnosu na test bez potvrđivanja. Ono što je bitno za ovaj sustav je da je ovaj način stabilan, jednako brz kao prvi kod većih podataka, a puno brži od prvog testa kod manjih podataka. Zapis podataka na lokalnoj mreži je kod malih podataka i dalje sličan brzini android memorije.



**Slika 4.11:** Usporedba brzine pisanja podataka sustava s minimalnim potvrđivanjem s lokalnim datotečnim sustavima



**Slika 4.12:** Usporedba svih testova u lokalnoj mreži



**Slika 4.13:** Usporedba svih testova u javnoj mreži

## 4.4. Korištenje sustava

Sustav se sastoji od poslužitelja i klijenta. Poslužiteljska aplikacija se koristi tako da se ukuca broj port-a koji će se koristiti za spajanje na poslužiteljsku aplikaciju nakon čega je više nije potrebno imati interakciju s aplikacijom, primjer se može vidjeti na slici 4.14. Kod klijentske aplikacije u potreblju je upisati više podataka potrebnih za sve testove koji se izvode, kao što je vidljivo na slici 4.15 to su redom:

1. IP adresa poslužitelja
2. pristupna točka poslužitelja
3. naziv vrste trajne memorije koja će se koristiti
4. lokacija na koju će se spremati testni podaci
5. ime testa
6. koristimo li sustav koji šalje podatke u zaglavljima (sustav bez potvrđivanja i sustav s minimalnim brojem potvrđivanja)
7. ako je prethodan odgovor potvrđan, treba li server slati potvrdu kada pročita cijeli podatak

Nakon unosa navedenih podataka aplikacija će izvršiti testove i zapisati rezultate u csv (eng. *Comma Separated Values*) datoteku.

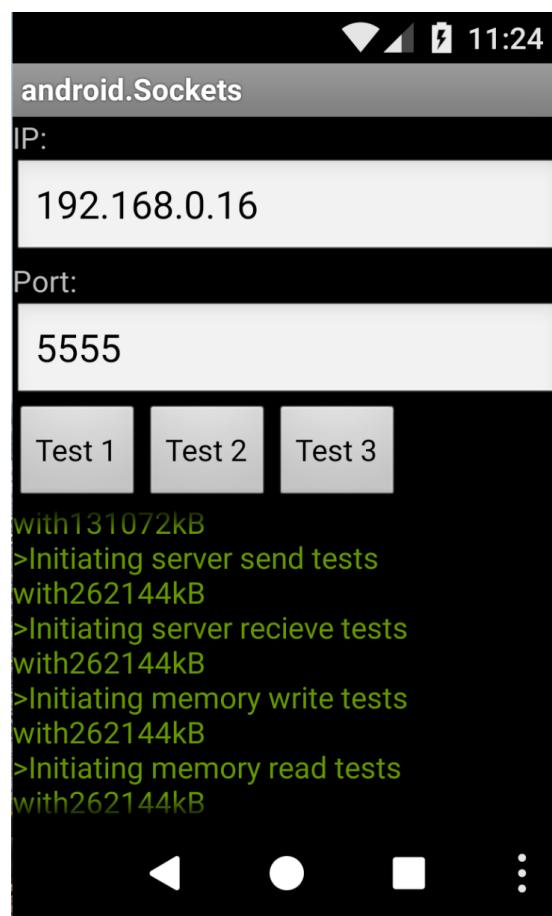
Android aplikacija sastoji se od dva polja u koja se upisuju IP i port poslužitelja, gumba za pokretanje testova i prozora s informacijama, kao što je vidljivo na slici 4.16. Za pokretanje testa unose se potrebni podaci i pritišće gumb koji odgovara testu koji se želi pokrenuti. Kada je test pokrenut odvijaju se jednake operacije kao kod klijenta na računalu.

```
Enter port number: 5555
Waiting for connection at: 192.168.0.48
Client connected on port 5555. Servicing requests.
```

Slika 4.14: prikaz terminala kod inicijalizacije poslužitelja

```
Enter host name: 192.168.0.41
Enter port number: 5555
Enter memory type name: ssd
Where do you want to save test files? .
Enter test name: test12
Use headers? (y/n)y
Use server check? (y/n)n
```

Slika 4.15: prikaz terminala prilikom upisa potrebnih podataka na klijentu



Slika 4.16: Zaslon android uređaja za vrijeme testa

## 5. Zaključak

U većini situacija zapisivanje i čitanje podataka brže je na trajnoj memoriji nego na radnu memoriju udaljenog računala. Jedina situacija kada se pokazalo da je brže poslati podatke na udaljeno računalo je kada je potrebno spremiti jako mali podatak, a nije potrebna povratna informacija od poslužitelja da je zapis uspio. Budući da kod malih podataka ne dolazi do pucanja veze s poslužiteljem, taj je pristup prihvatljiv ako je radnoj memoriji dovoljno držati samo manje podatke na drugoj memoriji. Iako postoji situacija u kojoj se isplati slati podatke na udaljeno računalo, ta situacija nije dovoljno učestala da bi je opravdali kao standardnu operaciju, pogotovo jer je korištenje lokalnog datotečnog sustava jednostavnije i pouzdanije rješenje. Jedino na android uređaju koji u pravilu imaju manju radnu memoriju i lokalnu memoriju, a zapis malih podataka na trajnu memoriju je relativno spor, ovakvo rješenje se isplati uzeti u obzir. Naravno jednom kada su podaci poslani na udaljeno računalo, potrebno ih je i dohvatiti, a tu se udaljeno računalo uvijek pokazalo kao sporije od trajne memorije.

# LITERATURA

- [1] Tvrđi disk. URL [https://hr.wikipedia.org/wiki/Tvrđi\\_disk](https://hr.wikipedia.org/wiki/Tvrđi_disk).
- [2] Java platform, api specification. URL <https://docs.oracle.com/javase/7/docs/api/index.html>.
- [3] The random access memory computer science essay. URL <https://www.ukessays.com/essays/computer-science/the-random-access-memory-computer-science-essay.php>.
- [4] Peter Haugen, Ian Myers, Bret Sadler, i John Whidden. A basic overview of commonly encountered types of random access memory (ram). URL <https://www.rose-hulman.edu/Class/ee/yoder/ece332/Papers/RAM%20Technologies.pdf>.
- [5] Joel Hruska. How do ssds work? URL <https://www.extremetech.com/extreme/210492-extremetech-explains-how-do-ssds-work>.
- [6] Limi Kalita. Socket programming. *International Journal of Computer Science and Information Technologies*, 5(3), 2014. URL <http://ijcsit.com/docs/Volume%205/vol5issue03/ijcsit20140503462.pdf>.
- [7] Jamie McKane. Ssd vs hdd – rand per gb price comparison, March 2018. URL <https://mybroadband.co.za/news/hardware/253373-ssd-vs-hdd-rand-per-gb-price-comparison.html>.
- [8] Bradley Mitchell. An overview of socket programming for computer networking. 2017. URL <https://www.lifewire.com/socket-programming-for-computer-networking-4056385>.
- [9] Chris Woodford. Hard drives. URL <https://www.explainthatstuff.com/harddrive.html>.

## **Usporedba brzine pristupa velikoj količini podataka smještenih u lokalnom datotečnom sustavu i udaljenoj radnoj memoriji**

### **Sažetak**

Glavne karakteristika radne memorije računala je ta da je brza, ali nije pogodna za trajno skladištenje podataka. Zbog toga, ali i zato što je skupa, to je memorija manjeg kapaciteta od trajne memorije. Ako lokalna radna memorija nema dovoljan kapacitet potrebno je podatke zapisati u trajnu memoriju ili poslati podatke na udaljeno računalo koje ih može čuvati spremne na svojoj radnoj memoriji. Najveća prednost udaljene radne memorije je ta što su ti podaci spremni za pristup većem broju računala. Nažalost pokazalo se kako je pristup podacima na radnoj memoriji udaljenog računala sporiji od pristupa na trajnoj memoriji, osim kod jako malih podataka kada nije potrebna potvrda od poslužitelja da su podaci uspješno pročitani. Zbog toga, ako se ti podaci koriste samo za radnje pojedinog računala, najbolje je koristiti SSD memoriju koja je jednako brza kod manjih podataka, a puno brža kod velikih podataka.

**Ključne riječi:** čvrsti disk, hdd, ssd, ssd memorija, mrežna priključna točka, radna memorija

## **Performance Evaluation of Accessing Large Datasets Using Local File System and Remote RAM**

### **Abstract**

Main characteristics of random access memory is that it is fast but not suitable for long term storage. For that reason, and it's price, such memory is of lower capacity than non-volatile memory storage. If local RAM doesn't have big enough capacity data must be stored in local memory or on remote device which can keep it ready in its random access memory. The biggest advantage of remote random access memory is that it can be used by multiple devices. Unfortunately it has been shown that data access to random access memory of remote computer is slower than accessing non-volatile memory, except with very small data when there is no need for server to confirm that data is successfully read. For those reasons, if data is used only for single device, it is best to use solid-state drive which is just as fast with small data, and much faster with big data.

**Keywords:** hard disk, hdd, ssd, Solid State Drive, Socket, Network Socket, random access memory, ram