

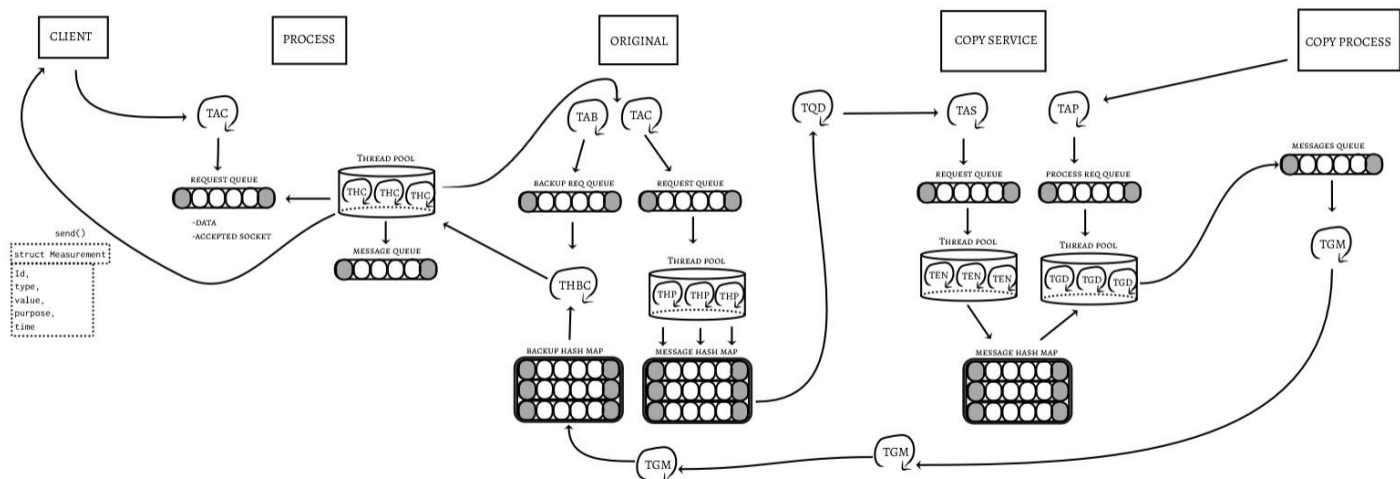
IKP – Replikacija

• UVOD

Bavimo se rešavanjem problema čuvanja i replikacije podataka. Servisi komuniciraju preko parova procesa za čuvanje podataka, koji se nalaze na dva računara, originalu i kopiji. Jedan ili više klijenata šalju merenje struje, napona ili snage u jednom delu mreže, određenom procesu na svom servisu, taj proces se kači na određeni red i smešta poruke u njega, odakle se dalje prosleđuju kopiji.

Cilj je obaviti pouzdanu isporuku podataka, u ovom slučaju merenja, sa originala na kopiju, gde se podaci pohranjuju u red za čuvanje do potrebnog backup-a.

• DIZAJN



Klijent šalje poruke(merenja) procesu na originalu. Na procesu se nalazi **TAC**(thread accept clients) koji zahteve smešta u red zahteva(implementiran kao kružni buffer struktura podataka kojeg čini pok. na niz socket-a). Thread pool koji broji 4 thread-a, preuzima zahtev sa porukom iz request queue-a i prosledjuje ga dalje Replication servisu, ako je aktivan, u suprotnom smesta podatak u red za poruke(koji je implementiran kao queue struktura podataka čija je vrednost tipa Measurement). Na originalu se nalazi **TAP**(thread accept process) koji u zavisnosti od prosledjenih parametara(a to je socket na koji je pristigao zahtev, imamo dva porta na kojima soketi osluškiju, za standardne poruke merenja i za klijentsko iniciranje backup-a) inicijalizuje socket i smešta ga u odredjeni red zahteva i obaveštava potrebnu nit za obradu zahteva. **THP**(thread handle processes) threadovi iz Thread poola služe za

obradu tih zahteva. Uzima se jedan thread iz thread pool-a, preuzima socket iz reda zahteva i prima poruku, ako red ne postoji, inicijalizuje ga, i poruku mapira po njenom ID-u na odgovarajući red u Hash mapi za privremeno skladištenje poruka.

TQD(thread queue delivery) pouzdano i paralelno šalje sve poruke iz redova na ReplicationService-u ka CopyService-u, za svaki uređaj posebnom konekcijom, brišući ih iz reda tek kad bude potvrđeno da su uspešno prosleđene.

TGM(thread get messages) služi za pokretanje backup-a, tj. oporavka podataka sa kopije i dodavanje merenja u heš mapu za backup poruka.

U slučaju da je klijent poslao poruku za backup, aktiviraće se **THBC**(thread handle backup to client) koji preuzima zahtev iz reda zahteva za backup od strane klijenta, zatim pristupa heš mapi za skladištenje podataka za backup i jednu po jednu poruku šalje nazad klijentu preko thread-a na procesu.

Što se tiče servisa na strani kopije imamo Acceptor Thread kojem se takodje preko parametara prosledjuje da li je zahtev stigao na port za servis ili proces. Na osnovu ishoda dodaje zahteve u odgovarajući red i obaveštava potrebne niti za dalje rukovanje porukama.

TEN(thread enqueue and notify) iz thread pool-a obavljaju obradu zahteva iz reda sa servisa i mapiraju poruku u mapu po ID-u merenja.

Threadovi iz thread pool-a **TGD**(thread get data from queue) pokreću procese za kopiranje, kada ih kondiciona variabla obavesti da su zahtevi od procesa primljeni, započinju vadenje poruka sa reda iz heš mape i prosledjuju odgovarajućem procesu.

Proces prima poruku od servisa za kopiranje i smešta je u red. Procesi se po pokretanju takodje povezuju na port koji služi za backup na servisu za kopiranje.

TGM(thread get messages) na procesu za kopiranje kada od servisa dobije zahtev za backup šalje podatke **TGM** thread-u na kopiji, koji prosledjuje originalu.

• STRUKTURE PODATAKA

• request_queue

Implementiran kao kružni bafer(ring buffer). Kružni bafer radi tako što koristi fiksni niz za skladištenje elemenata, ali koristi pokazivače (front i rear) i modularnu aritmetiku da simulira beskonačni red – kada se dodje do kraja niza, nastavlja se od početka. Služi za primanje socket-a i njegovo smeštanje u niz, čuva pokazivač na niz socket-a. Klasičan red sa pomeranjem elemenata je neefikasan ($O(n)$ za svaki dequeue).

Kružni bafer koristi fiksni prostor i $O(1)$ je za sve operacije.

- queue

Red implementiran preko povezane liste. Svaki čvor čuva jedan objekat tipa `Measurement` i pokazivač na sledeći čvor. Red je implementiran pomoću pokazivača na prvi (`head`) i poslednji element (`tail`) kao i broja elemenata (`size`). Ova implementacija je idealna za dinamičke redove (nemaju fiksnu veličinu) sa jednostavnim, efikasnim operacijama na početku i kraju reda.

- unoredred_map

Ovo je implementacija heš tabele sa ulančanim listama, gde je vrednost za svaki ključ zapravo red (`queue`). Svaki par čuva *ključ*, njegovu *vrednost* i *sledeći par* ako dođe do sudara (kolizije) na istoj heš lokaciji.

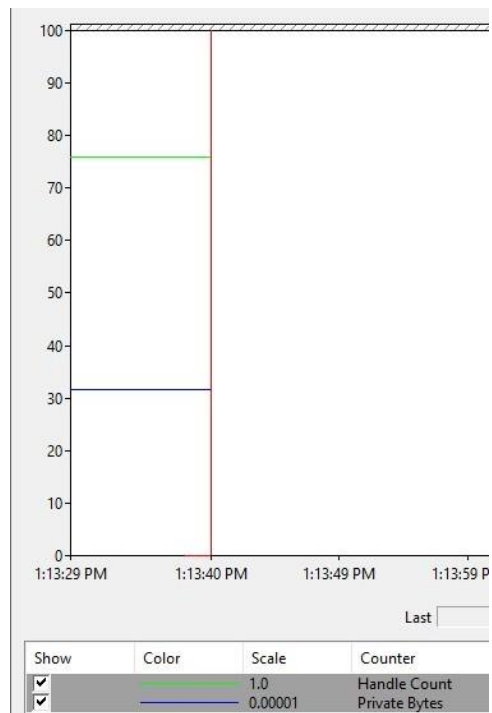
```
typedef struct {  
    KeyValuePair** buckets; // Niz pokazivača na ulančane liste parova  
    size_t capacity;         // Broj bucket-a tj. mesta u heš tabeli  
    size_t size;             // Broj ukupnih elemenata u mapi  
} unordered_map;
```

Na servisima za replikaciju gde imamo konekcije od vise procesa i klijenata hash map-a se čini kao najlogičniji izbor, a pogodna je zbog svoje fleksibilnosti, brzog pristupa i rešavanja kolizije ulančavanjem.

- REZULTATI TESTIRANJA

Testovi su izvršeni nad Replication Servisom sa odgovarajućim dijagramima:

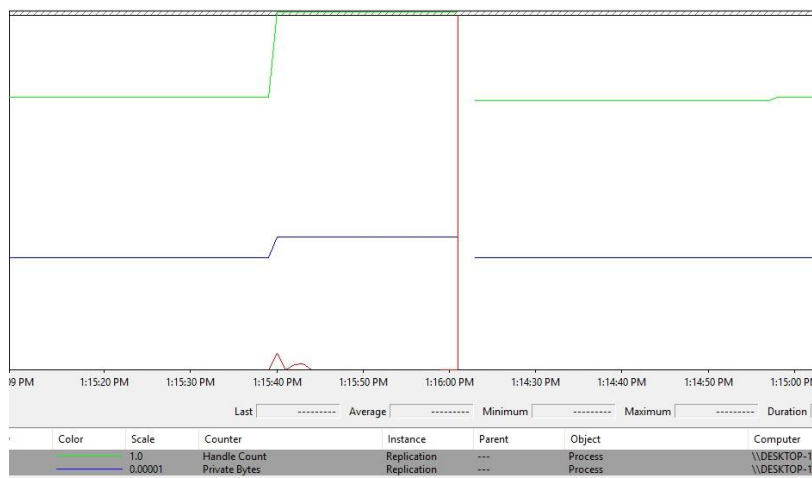
1. Pre inicijalizacije potrebnih struktura (redova, mapa...):



- Processor Time;
- Private Bytes;
- Handle Count;

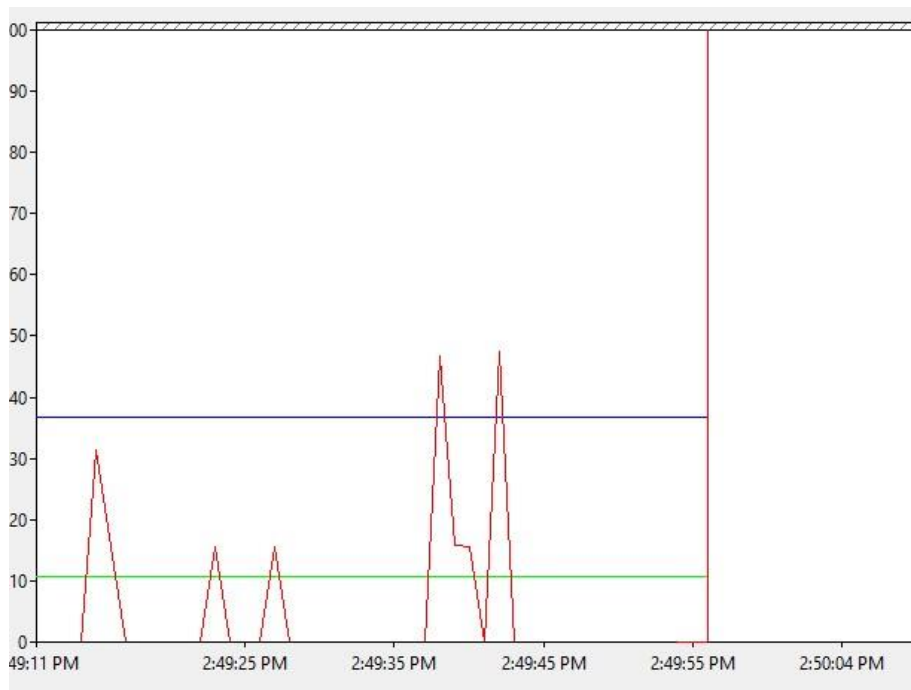
2. Nakon uspostavljanja konekcije i kreiranja handle-ova:

	ID	Time	Allocations (Diff)	Heap Size (Diff)	Label
	1	1.85s	1,486 (n/a)	364.31 KB (n/a)	
	2	1.86s	1,492 (+6 ↑)	365.16 KB (+0.85 KB ↑)	
	3	5.66s	1,816 (+324 ↑)	503.48 KB (+138.31 KB ↑)	



Zbog alociranja memorije procenat rada procesora, kao i private bytes su skočili, kao i broj handle-ova za prijem i obradu podataka.

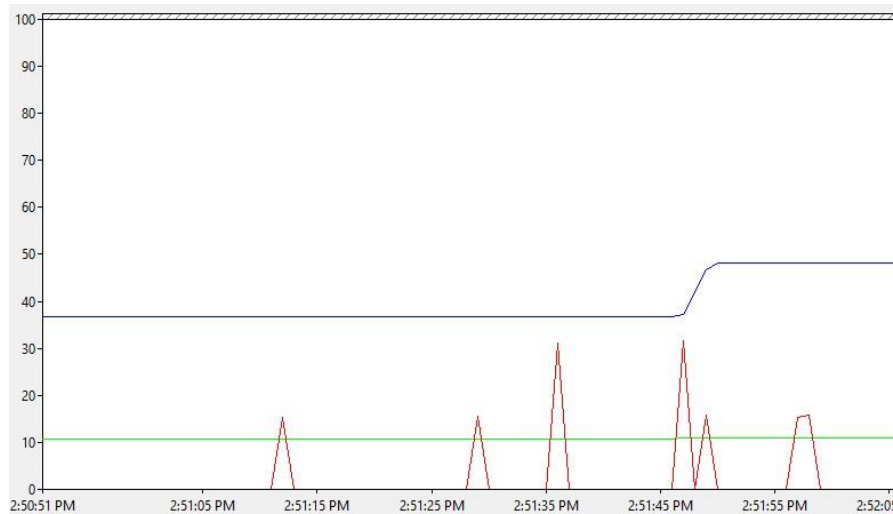
3. Klijent poslao 10 000 poruka:



Na dijagramu vidimo oscilacije u radu procesora prilikom primanja i prosledjivanja velikog broja podataka.

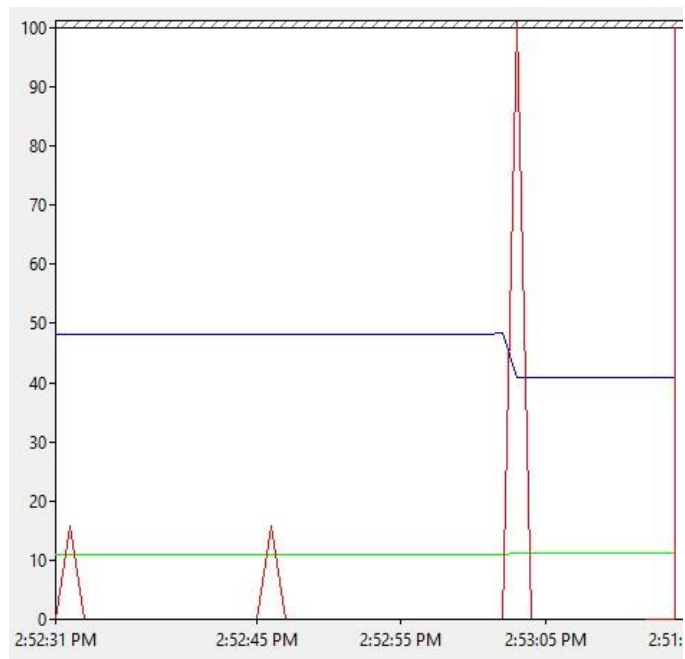
4. Backup podataka sa kopije na original:

	ID	Time	Allocations (Diff)	Heap Size (Diff)	Label
	1	1.85s	1,486 (n/a)	364.31 KB (n/a)	
	2	1.86s	1,492 (+6 ↑)	365.16 KB (+0.85 KB ↑)	
	3	5.66s	1,816 (+324 ↑)	503.48 KB (+138.31 KB ↑)	
	5	127.78s	1,888 (+72 ↑)	514.72 KB (+11.25 KB ↑)	
	7	170.88s	11,896 (+10,008 ↑)	1,335.79 KB (+821.07 KB ↑)	



Private Bytes se povećao jer su se oporavljeni podaci pohranili u red za backup.


5. Vraćanje podataka klijentu(Client Initiated Backup):



	ID	Time	Allocations (Diff)	Heap Size (Diff)	Label
	1	1.85s	1,486 (n/a)	364.31 KB (n/a)	
	2	1.86s	1,492 (+6 ↑)	365.16 KB (+0.85 KB ↑)	
	3	5.66s	1,816 (+324 ↑)	503.48 KB (+138.31 KB ↑)	
	5	127.78s	1,888 (+72 ↑)	514.72 KB (+11.25 KB ↑)	
	7	170.88s	11,896 (+10,008 ↑)	1,335.79 KB (+821.07 KB ↑)	
	8	238.93s	1,900 (-9,996 ↓)	516.22 KB (-819.58 KB ↓)	

Ponovo vidmo porast opterećenja rada procesora usled uzimanja velikog broja poruka iz reda i slanja, private bytes opada, što znači da se nakon slanja podataka memorija uspešno oslobodila.

6. Finalni sled događaja na heap-u nakon zatvaranja konekcija i prekida servisa:

	ID	Time	Allocations (Diff)	Heap Size (Diff)	Label	
	1	1.85s	1,486 (n/a)	364.31 KB (n/a)		
	2	1.86s	1,492 (+6 ↑)	365.16 KB (+0.85 KB ↑)		
	3	5.66s	1,816 (+324 ↑)	503.48 KB (+138.31 KB ↑)		
	5	127.78s	1,888 (+72 ↑)	514.72 KB (+11.25 KB ↑)		
	7	170.88s	11,896 (+10,008 ↑)	1,335.79 KB (+821.07 KB ↑)		
	8	238.93s	1,900 (-9,996 ↓)	516.22 KB (-819.58 KB ↓)		
	9	282.28s	1,640 (-260 ↓)	413.11 KB (-103.10 KB ↓)		

— memorija koja se alocira tokom najzahtevnijih operacija (popunjavanje reda, backup) se kasnije oslobodi (heap size pada praktično na vrednost sa početka).

Iz ovoga možemo zaključiti da nema curenja memorije jer bi u suprotnom heap size rastao nakon svakog ciklusa i nikad se ne bi vratio na približnu osnovnu vrednost.