Load Balancer

Dokumentacija

Uvod

Kod klasicne klijent – server arhitekture javlja se problem usluzivanja vise klijenata. Jedan server je odgovoran za primanje i obradu zahteva cime dolazi do problema velikog cekanja sa strane klijenata koji cekaju uslugu i velikog opterecivanja serverskih resursa.

Cilj implementacije Load Balancera je poboljsavanje raspodele klijentskih zahteva na vise razlicitih procesa, koji se nazivaju workeri, optimizacijom raspolozivih resursa i smanjenjem vremena odziva servera bez opterecivanja bilo kojeg resursa. Load Balancing-om se povecava pouzdanost i dostupnost kroz koriscenje vise workera.

Cilj zadatka je razvijanje komponente koja vrsi dinamicko balansiranje opterecenja time sto ce kroz ispitivanje i kontrolu stanja reda zahteva pokretati i gasiti workere. Uvodjenjem komponente Load Balancer, server ne zavisi vise od dostupnosti samo jednog procesa vec postoji vise procesa koji se aktiviraju u zavisnosti od prometa i dostupnosti resursa.

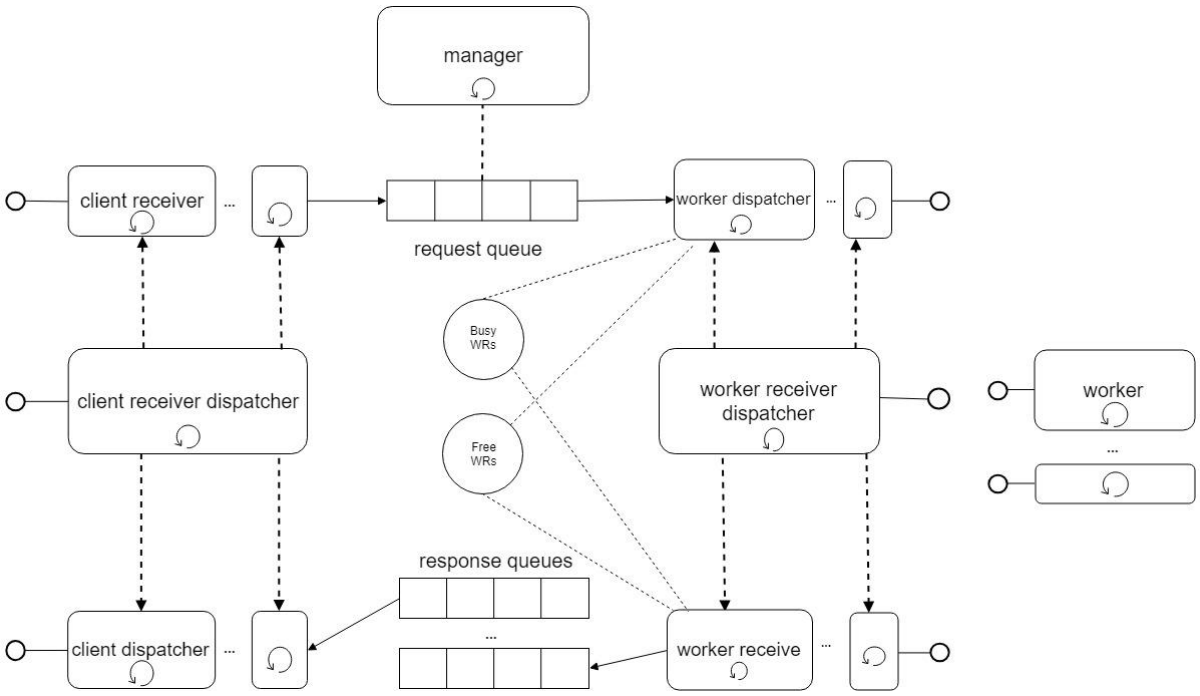
Dizajn

Glavna ideja dizajna komponente Load Balancer-a je maksimalno iskoriscenje procesorske moci i mrezne propusnosti servera.

Sa stanovista procesora dizajn se trudi da maksimalno iskoristi moguci paralelizam i konkurenciju time sto se cela komponenta svodi na nekoliko glavnih thread-ova (manjih komponenti) koje dodatno pokrecu svoje thread-ove za slanje i primanje podataka.

Iz pogleda propusnosti mreze cinjenica da svaki klijent i worker imaju poseban thread omogucuje maksimalnu mogucu prolaznost bez nepotrebnog cekanja. Koriscenjem TCP neblokirajucih socketa i funkcija za manipulaciju istim izbacuje se potreba za polling modelima i nepotrebnim cekanjem.

Zbog slucaja slabe propusnosti mreze izmedju klijenta i servera, za svakog klijenta se otvara poseban red kako bi klijenti primili obrade svojih zahteva na pouzdan nacin.



Dijagram 1. Dizajn zadatog Load Balancing projekta

Load Balancer

`start_client_receiver_dispatcher`

Thread koji osluskuje i prima nove klijente u neblokirajucem rezimu, koristeci TCP protokol zapocinjuci nove threadove `start_client_receive` i `start_client_dispatch`, inicijalizuje za svakog klijenta poseban red u kome se primaju odgovori od workera.

`start_client_receive`

Thread koji prima zahteve od konektovanih klijenata pakuje ih u `Request` strukturu i pushuje u red sa zadacima.

`start_client_dispatch`

Thread koji skida poruke sa reda odgovora i salje ih konektovanim klijentima i pri zatvorenoj konekciji sa klijentom unistava klijentov red odgovora.

`start_manager`

Thread koji inicijalizuje procese worker-a i vrsi proveru nad redom. U slucaju da je red popunjen vise od 70% pokrenuce novog worker-a. U slucaju da je red popunjen manje od 30% na odredjenom intervalu gasice slobodne workere kojima ima pristup preko `WorkerList`-e.

`start_worker_receiver_dispatcher`

Thread koji osluskuje i prima nove worker-e u neblokirajucem rezimu, koristeci TCP protokol zapocinjuci nove threadove `start_worker_receive` i `start_worker_dispatch`,

`start_worker_receive`

Thread koji prima odgovore od worker-a i stavlja ih u odgovarajuci klijentov red odgovora.

`start_worker_dispatch`

Thread koji salje zahteve inicijalizovanim worker-ima preko socket-a kojeg dobija iz liste slobodnih worker-a. Pop-uje worker-a iz liste i funkcija `worker_list_pop` vraca socket na kome je konektovan worker i salje mu zahtev.

Klijentska strana

`start_receiver`

Thread koji zapocinje primanje odgovora od server koristeci konektovani socket.

`start_sender`

Thread koji prima odgovore od servera koristeci konektovani socket.

Worker

Worker je komponenta koja radi u zasebnom procesu. Njegova uloga je da prima klijentske zahteve od servera i da ih obradi. Obrada zahteva se simulira uspavljivanjem niti na predodredjeno vreme. Nakon simulirane obrade zahteva worker vraca odgovor LoadBalancer-u I njegovom start_worker_recieve thread koji osluskuje odgovore kako bi ga on dalje prosledio klijentu.

Zajednicka biblioteka

Tcp_helper biblioteka je staticka biblioteka koja sadrzi generalne funkcije za konektovanje, slanje i primanje poruka. Toj biblioteci imaju pristup svi projekti. U njoj je podrzana logika za postavljanje socket-a na neblokirajuci mod koriscenjem `ioctlsocket` funkcije. Funkcije `select_for_receive` i `select_for_send` pruzaju podrsku u vidu set-a koji sadrzi socket-e. Dok funkcije `receive` i `send` podrzavaju enkapsulaciju poruke i njeno slanje.

Strukture podataka

Opis korišćenih struktura podataka(Razlozi zbog kojih su izabrane baš te strukture podataka):

Worker lista slobodnih workera je implementirana kao FIFO.

```
typedef struct request_t {
	int dataLength
	char *data;
} Request;
```

Struktura klijentskog zahteva

Struktura klijentskog zahteva koju pravi `start_client_receive` pri dobijanju zahteva od klijenta. Sadrzi duzinu poruke i njen sadrzaj. Nakon kreiranja, tu strukturu `start_client_receive` push-uje u red zahteva. Takvu vrstu strukture koristi i `Response` struktura cija je jedina razlika to sto je `start_worker_receive` smesta u poseban red odgovora koji se nakon toga salje klijentu.

```
typedef struct request_queue_t {
	int tail;
	int head;
	int count;
	Request requests[REQUEST_QUEUE_CAPACITY];
	HANDLE signalFull;
	HANDLE signalEmpty;
	CRITICAL_SECTION lock;
} RequestQueue;
```

Izgled kruznog bafera

Struktura kruznog bafera koja sluzi za kontrolisanje i navigaciju samog kruznog bafera. `Tail` je indeksni pokazivac koji pokazuje na kraj popunjenog dela reda. `Head` je indeksni pokazivac koji pokazuje na pocetak popunjenog dela reda. `Count` je brojac zauzetih elemenata kojem pristupa manager radi provere pri kreiranju i gasenju worker-a. Niz `requests`

Opis i pojašnjenje semantike podataka koje sadrže strukture

Strukura WR liste:

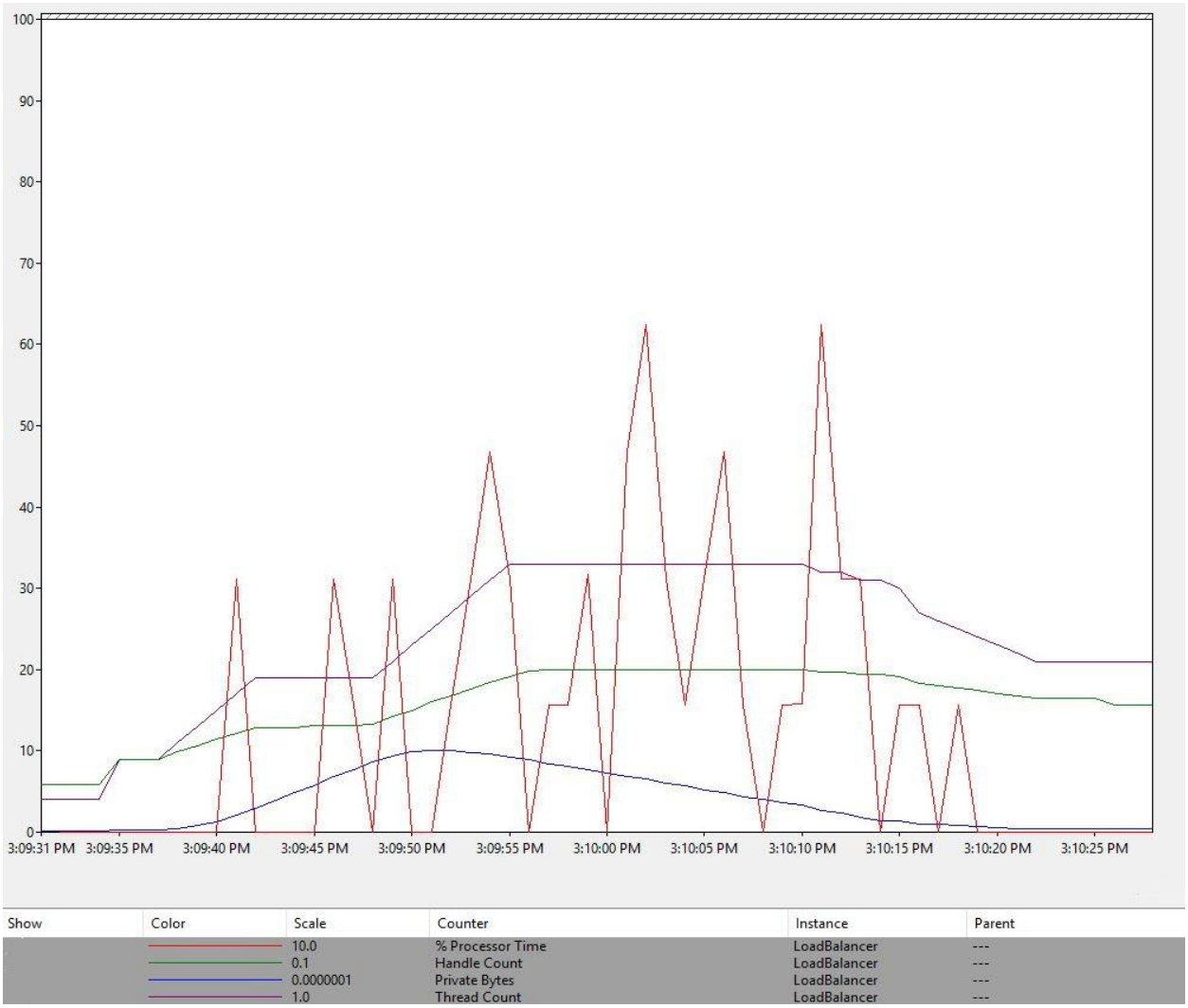
```
typedef struct worker_list_t
{
	p_Worker head;			///< Pokazivac na pocetak liste
	p_Worker tail;			///< Pokazivac na kraj liste
	int count;			///< Brojac elemenata u listi
	HANDLE signalEmpty;		///< Semafor
	CRITICAL_SECTION lock;		///<
} WorkerList;
```

Rezultati testiranja

Kod testiranja komponente dolazi do pitanja kako precizno konfigurisati: velicinu memorijskog prostora za poruke klijenata, maksimalan broj workera, maksimalan broj poruka u redu cekanja i slicno.

Za ovaj stress test je izabrana sledeca konfiguracija:

- 5 Klijenata koji istovremeno salju 20 zahteva i ocekuju 20 odgovora.
- Brzina slanja je 2 MB u sekundi.
- Load Balancer sa redom zahteva i klijentskim redovima velicine 100.
- Kapacitet workera je 20 i provera zagusenja se izvrsava svake sekunde.
- Velicina mreznih bafera je 1 MB.



Rezultati stress testa

Zakljucak

Test je izvrsen u trajanju od priblizno 60 sekundi sa tim da je od trenutna prvog pristiglog klijentskog zahteva do poslednjeg poslatog odgovora klijentu proslo priblizno 40 sekundi. Sa slike se moze primetiti da je nakon 15 sekunda menadzerska komponenta Load Balancer-a detektovala zagusenje reda i pocela da pokrece nove worker-e

Potencijalna unapredjenja