Miloš Savić

YAMOS: *multi-computer* operativni sistem

YAMOS je modifikacija openMosix multi-computer operativnog sistema sa osnovnom idejom da umesto celog procesa migriraju samo njegovi delovi, odnosno niti (engl. threads). Da bi se sistem realizovao, nadograđen je postojeći threading model LINUX kernela, implementiran novi sistem za migraciju procesa i realizovan novi model mrežnih semafora. Ceo koncept omogućava manji mrežni saobraćaj i optimalnije iskorišćenje resursa klastera.

Uvod

Moderni *multi-computer* operativni sistemi se sastoje iz skupa algoritama koji omogućavaju deljenje resursa u CC (*cluster computing*) sistemu. Osnova jedne ovakve tehnologije je mogućnost da umreženi računari (radne stanice i serveri, uključujući i višeprocesorske računare) rade kao jedinstven sistem (Barak, La'adan 1998).

Ideja je da se alati koji omogućuju CC implementiraju direktno u kernelu operativnog sistema. Dizajnirani su algoritmi koji na osnovu varijacija u korišćenju resursa između nodova, prebacuju proces sa jednog noda klastera na drugi (Barak, La'adan 1998). Transfer procesa sa jednog noda na drugi je poznat kao migracija procesa. *Multi-computer* operativni sistemi se sastoje iz dva dela: mehanizma za migraciju procesa i algoritma za alokaciju resursa klastera (Barak *et al*. 2000). Do sada je poznato više razrada ove ideje namenjene za Linux operativni sistem (Mosix, openMosix, Beowulf), međutim, u svim ovim rešenjima migriraju procesi a ne threadovi, što je posledica threading modela koji koristi operativni sistem LINUX (Beck *et al* 1996).

Cilj rada je dobiti što efikasniji CC operativni sistem, sa što manjim mrežnim protokom. Veoma jednostavnom izmenom threading modela i postojećih algoritama za migraciju procesa može se postići da umesto celog procesa, migriraju samo delovi procesa koji pripadaju jednom threadu. U daljem radu se diskutuje način na koji je to postignuto i potrebne adaptacije Linux i openMosix kernela u novonastaloj situaciji.

Nadograđeni threading model

Da bi se omogućila migracija threadova, morao je biti promenjen postojeći threading model LINUX kernela. Novi model omogućava kernelu da održava specijalnu tabelu, koja se zove tabela threadova, u kojoj se nalaze pointeri na privatne resurse threada.

Tabela threadova sadrži sledeće podatke: identifikator threada – podatak koji je jedinstven za svaki thread; pointer na slot u tabeli procesa, gde se nalazi dotični thread; adrese početka i kraja svakog segmenta threada (code, data i stack), pointeri na memorijske mape koje su privatne za dati thread i niz pointera na listu otvorenih datoteka, koje je otvorio dati thread. Identifikator threada, po ugledu na Process ID (PID), dobija se pri kreiranju threada i služi za njegovo referenciranje. Migracija threadova se vrši na osnovu ove tabele. Modifikovani su sistemski pozivi sys_clone() i sys_exit(), koji odvajaju/oslobađaju slot u tabeli thredova za kreirani/uništeni thread, kao i sistemski pozivi sys_brk() i sys_open(), koji menjaju podatke u tabeli thredova.

Migracija thredova

Svaki thread ima tzv. unique-home node (UHN), tj nod u klasteru gde je kreiran. Thread koji migrira na drugi node (remote), koristi resurse toga noda kada god je to moguće, ali vrši interakciju sa svojim korisničkim okruženjem preko UHN-a. Migracija

Miloš Savić (1984), Valjevo, Maljenska bb, učenik 3. razreda Valjevske gimnazije

thredova je podeljena na dva konteksta: *user* i *system* kontekst. *System* kontekst je UHN zavisan i ne može migrirati.

User kontekst threada, za razliku od ostalih multicomputer operativnih sistema, migrira ceo, zato što je thread logička celina i vrlo je verovatno da će se ceo izvršiti od početka do kraja. Ostali delovi procesa koje thread može koristiti migriraju po *lazy* principu (na zahtev, odnosno kada je to potrebno) jer pripadaju istom virtuelnom prostoru. To je regulisano NVMM-om, odnosno *network virtual memory managerom*. NVMM je nastao kao modifikacija postojećeg LINUX *memory manager*-a, tako da kada se dogodi *page fault,* threadu koji se ne izvršava na svom UHN-u, migrira ona stranica za čiju virtuelnu adresu se dogodio *page fault*. Na taj način omogućena je migracija threadova, bez migracije celog procesa kome thread pripada. Posledica toga je optimalan mrežni protok, te samim tim klaster postaje "dinamičniji", što teži boljem iskorišćenju samih resursa klastera.

Migracija thredova omogućena je rutinama koje su implementirane u okviru nadograđenog threading modela. Migracija je apstraktno podeljena na dva nivoa: slanje i primanje *user* konteksta. Fizičku migraciju obavlja *daemon*, koji se sastoji iz servera i klijenta. Server radi kao pozadinski proces (implementiran kao *multithreaded* server), prima user context i restaurira ga na nodu na kome se izvršava. Server zauzima određeni port na kome glavni thread (master) čeka konekcije (po defaultu to je port 5959). Pri svakoj konekciji na server, tj. kada master thread primi asinhronu poruku za prijem konekcije, master thread kreira thread potomak, startuje ga, i, u specijalnoj strukturi podataka koja se zove lista zahteva (engl. request list), pamti novokreirani *socket* kojim thread potomak obavlja transfer *user* konteksta sa klijentom i identifikator thread potomka. Master thread nastavlja da čeka konekcije, tako da je teorijski moguć veliki broj konekcija. Kada master thread dobije poruku da je thread potomak završio transfer podataka, stopira i uništava thread potomak i briše njegove informacije iz liste zahteva. *Daemon* koristi *precopying*, što znači da thread nastavlja da se izvršava na *remote* nodu, dok se vrši asinhroni transfer njegove slike. Pre nego što počne sam transfer threada, thread se mora stopirati (zamrznuti) na deputy nodu. Klijent, sliku threada razbija u pakete fiksne veličine, koja iznosi koliko i veličina same stranice (4 kb na 386 platformi) i šalje ih serveru. Kada server primi određeni broj paketa koji sadrže *code, data i stack* segment (po defaultu to je 6), thread se startuje (odmrzava) na remote mašini. Svaki sledeći paket server uključuje u adresni prostor datog threada. Kada dođe do potrebe za *lazy-copying*om neke stranice, klijent inicira slanje iste, server prima stranicu i restaurira je u kontekst dotičnog threada.

Sinhronizacija threadova

Koncept migracije thredova nosi sa sobom i problem sinhronizacije thredova. U skladu sa tim, implemenirana je mrežna verzija semafora. Sastoji se iz servera, koji se nalazi na jednom od nodova, i klijenata koji se nalaze na svim nodovima u okviru klastera. Implementirane su Dajkstrine P i V operacije koje se izvršavaju na serveru. Server je implementiran kao pozadinski proces. U jednom vremenskom trenutku, server može opsluživati samo jednog klijenta. Ostali klijenti se stavljaju u red čekanja; na taj način je omogućena atomičnost P i V operacija. Operacije nad semaforima su: kreiranje i uništavanje semafora, P i V operacije. Klijent šalje zahtev za operacijom, server je izvršava i vraća rezultat, koji je za prve dve operacije identifikator semafora, dok za druge dve informacija da li thread koji je generisao dotične operacije mora biti uspavan, ili može probuditi neki drugi thread iz reda čekanja na semafor. Modifikovan je *scheduler* LINUX kernela, koji sada može na osnovu klijent procesa koji obavlja komunikaciju sa serverom na kome se izvršavaju P i V operacije uspavati/probuditi proces, na osnovu relzultata koji su posledica izvršvanja P i V operacija na serveru.

Zaključak

U ovom radu dat je novi pristup u implementaciji jednog multi-computer operativnog sistema. Mogućnost koju nudi YAMOS je migracija threadova, a ne celih procesa. Time se redukuje mrežni saobraćaj, u slučajevima kada threadovi u najvećoj meri funkcionišu samostalno. U slučajevima kada su threadovi jako međuzavisni i dele velike nesusedne delove adresnog prostora, dolazi do usporenja (ping-pong efekat). Za razliku od openMosix-a, programi koji se izvršavaju na klasteru moraju biti modifikovani da bi se mogli uklasteriti (zbog problema sinhronizacije), tj. svi deljivi delovi se moraju obezbediti mrežnim semaforima.

Literatura

Barak, La'adan. 1998. *The MOSIX Multicomputer operating system for high performance cluster computing*

Barak, La'adan, Shiloh. 2000. *Scalable Cluster Computing with Mosix for Linux*

Beck, Bohme, Dziadzk, Kunitz, Magnus, Verworner. 1996. *Linux kernel internals*. Prentice Hall.

Miloš Savić

YAMOS: multi-computer operating system

YAMOS is a modification of the openMosix multi-computer operating system based on the idea of migrating threads instead of processes. In order to realize this concept, the threading model of the LINUX kernel is changed, a new process migration system is implemented and a new model of network semaphores is realized.

The new threading model enables kernel to keep a special table called thread table in order to bind private resources to a thread. The thread migration system uses this table when thread migrates to another node in a cluster. When a thread migrates, in the first migration only resources which are private to the thread are being transfered. Other resources migrate on demand which is regulated by a NVMM.

The NVMM is a modification of Linux memory manager with the main task to transfer pages at a time when the page fault occurs. The physical transfer of memory is done by a daemon which is implemented in the client/server manner. The new version of semaphores enables centralized P and V operations which can be applied to all threads in the cluster. The whole concept enables reduced network traffic and the better usage of cluster resources