Projet PSAR : Répartiteur de charge

Encadrant : **Pierre SENS**

Étudiants: Momar TOURÉ
Richard UNG

Année 2020/2021

Sommaire

Introduction

- Sujet
- Objectifs

L’équilibrage de charge désigne le procédé par lequel on distribue une charge entre plusieurs serveurs appartenant à un groupe de machines afin de pouvoir optimiser les performances et raccourcir les temps de réponses.

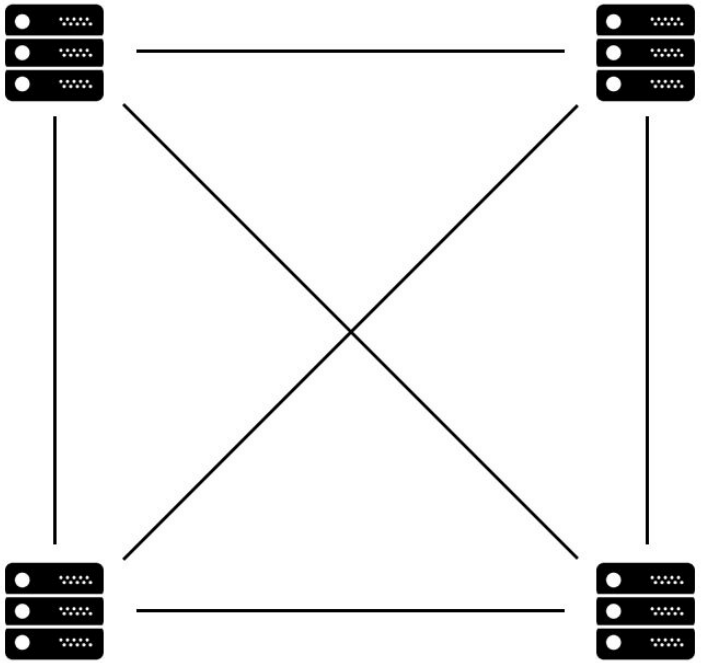
Introduction

- Sujet
  - Objectifs

- Evaluation de la charge de chaque machine
- Echange d’informations entre les machines
- Placement des charges dans le réseau
- Gestion de la surcharge/sous charge du réseau
- Implémentation des commandes

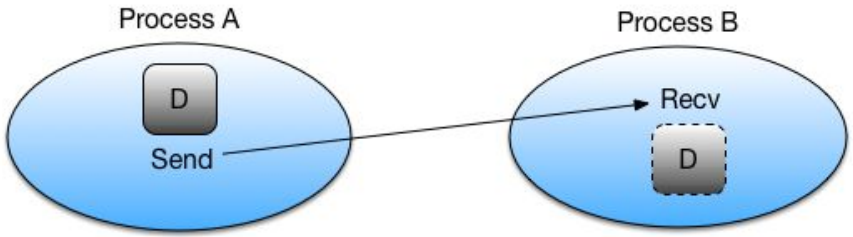
Architecture du réseau

- Structure du réseau
- Communication des machines



Architecture du réseau

- Structure du réseau
  - Communication des machines



Gestion des machines

- Structure interne d’une machine
- Table des processus
- Matrice de sauvegarde

- Table de participants
- Table des charges
- Table des processus
- Matrice de sauvegarde

Gestion des machines

- Structure interne d'une machine
  - Table des processus
- Matrice de sauvegarde

```
struct process{
        pid_t pid;
        int gpid;
        char* cmd;
}process[PROCESS_SIZE];
```

Gestion des machines

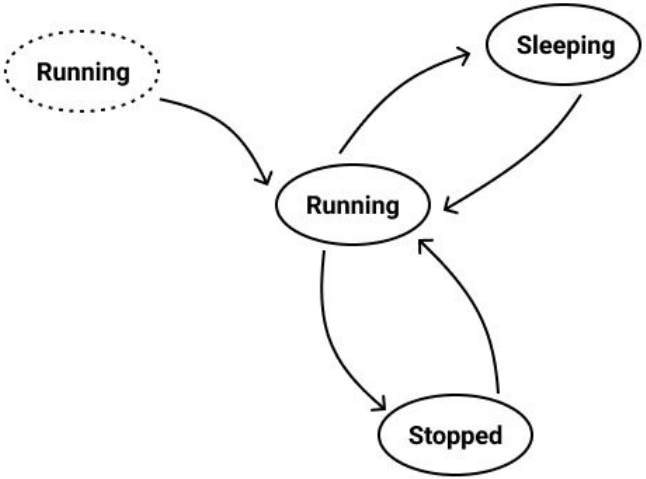
- Structure interne d'une machine
- table des processus
  - matrice de sauvegarde

```
int machines[NB_MACH][PROCESS_SIZE]
```

Gestion des charges

- Définition
- Surcharge
- Sous charge

```
$ cat /proc/loadavg
    0.52 0.50 0.54 1/313 13261
```



Gestion des charges

- Définition
  - Surcharge
- Sous charge

```
int surcharge(){

      Si participant:

            Calcul de la charge globale

            Si > 70% charge globale:

                  Récupère l'ensemble des
      machines de charge min

                  Si id de la machine appartient
      à l'ensemble de charge min:

                        Ajoute une machine

                  Sinon:

                        Transfert d'un processus
      à l'une des machines de charge min

}
```

Gestion des charges

- Définition
- Surcharge
  - Sous charge

```
int souscharge(){

      Si participant:

            Calcul de la charge globale

            Si < 30% charge globale:

                  Si il y a au moins 2 machines
      participantes dans le réseau:

                        Parcours de la table des charges

                              Si la première machine en
      sous charge que l'on trouve est soi-même:

                                    Retire la machine

                                    Envoie tous ses
      processus à la prochaine machine avec le moins
      de charge

}
```

Commande

- gstart
- gps
- gkill

```
void gstart(char * args[], int gpid, int indice_process){
        int pid = fork();
        if(pid==0){
                int i = 0;
                /* Le processus fils exécute la commande args[0] */
                if(!execvp(args[0],args)){
                perror("gstart : execvp failed\n");
                exit(0);
        }
}
   /* Le père enregistre les informations du fils :
   *  - identifiant du processus (locale à la machine)
   *  - identifiant globale du processus (globale au réseau)
   *  - la commande
   */
   (process + indice_process)->pid = pid;
   (process + indice_process)->gpid = gpid;
   (process + indice_process)->cmd = strdup(args[0]);
}
```

Commande

- gstart
  - gps
- gkill

```
void gps(int option){
    int p;
    int uid = getuid();
    //affichage du tableau process local de la machine
    if(option == 0){ // sans option
            // affiche tous les processus de sa table des processus
            for(p = 0; p < PROCESS_SIZE; p++){
                    if(process[p].pid != 0){
                            printf("%d\t%d\t%s\n", process[p].pid,
                                    process[p].gpid,process[p].cmd);
                    }
            }
    }else{ // format long car option -l
            //(noms executable, machine, uid, éventuellement statistiques
            d'utilisation CPU, mémoire)
            for(p = 0; p < PROCESS_SIZE; p++){
                    if(process[p].pid != 0){ // Les cases non instancié sont
                    ignorées
                    printf("%s\t%d\t%d\t%d\t%s\n",hostname, uid,
                    process[p].pid, process[p].gpid,process[p].cmd);
            }
```

Commande

- gstart
- gps
  - gkill

```
void gkill(int signal, int pid, int gpid, int p){
    char kill[20];
    // Lance la fct systeme kill
    printf("je dois kill le pid %d\n",pid);
    sprintf(kill, "kill -%d %d",signal, pid);
    printf("kill = %s\n", kill);
    system(kill);

    // On retire le processus de sa table de processus
    (process + p)->pid = 0;
    (process + p)->gpid = 0;
    (process + p)->cmd = NULL;

    // On prépare le message d'envoye
    int gkill_gpid[2];
    gkill_gpid[0] = gpid;
    gkill_gpid[1] = p;

    // On envoie le gpid et son indice dans la table de chaque machine
    // pour que chaque participant le retire
    for(int i = 1; i < nb_proc; i++){
       if((i != rank) && (tab_participe[i])){
          MPI_Send(gkill_gpid, 2, MPI_INT, i, TAG_GKILL_GPID,
            MPI_COMM_WORLD);
       }
    }
```

Difficultés rencontrées

- Sockets
- Installation du réseau
- Insertion/Retrait

Difficultés rencontrées

- Sockets
  - Installation du réseau
- Insertion/Retrait

- ssh-keygen -t rsa
- cd $HOME/.ssh
- cp id_rsa.pub authorized_keys
- eval `ssh-agent`
- ssh-add \$HOME/.ssh/id_rsa

Difficultés rencontrées

- Sockets
- Installation du réseau
  - Insertion/Retrait

Conclusion