Le Poller

Sommaire

Rôle

Le démon Poller exécute les sondes telles que planifiées par le ou les Schedulers.

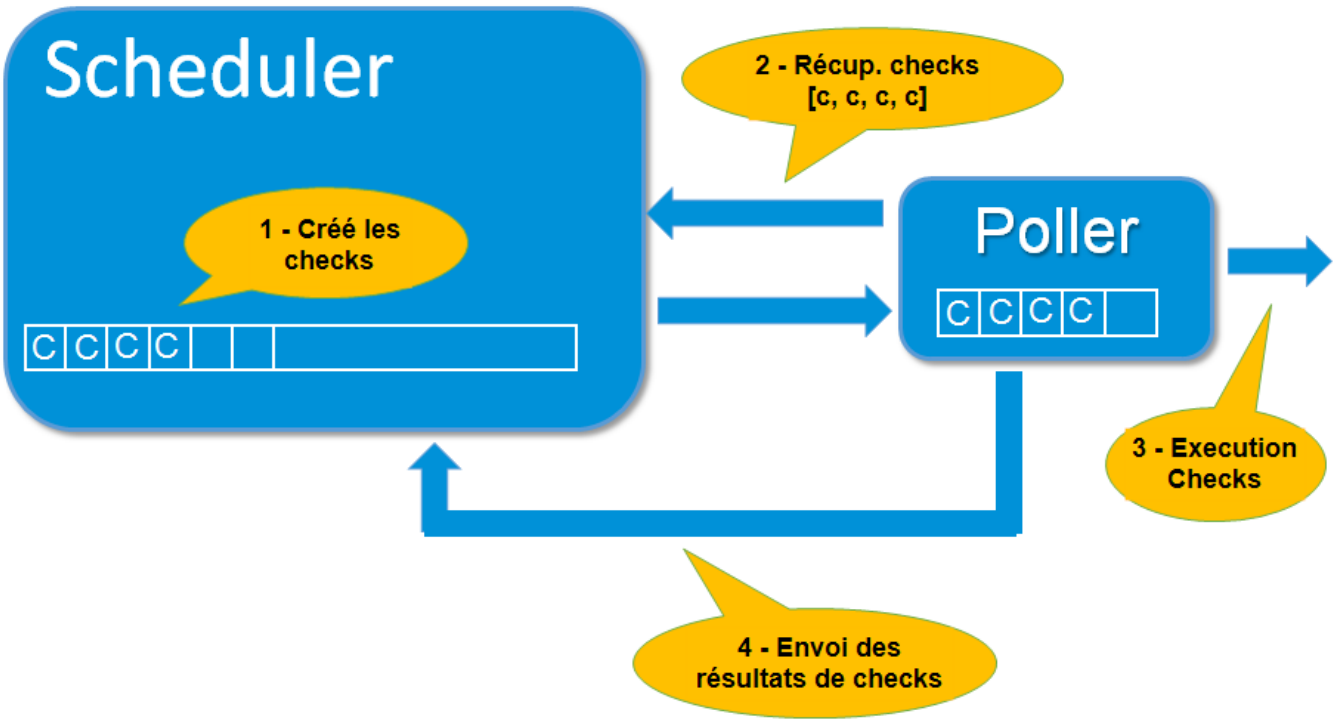
Il se connecte au(x) Scheduler(s) de son royaume et récupère les checks pour exécuter les commandes/sondes. Quand la commande est terminée, le retour de la sonde est envoyé aux Schedulers. Les Pollers peuvent être définis pour des vérifications spécifiques dans des environnements spécifiques (ex. Windows versus Unix, client A versus client B, DMZ).

Il peut également y avoir plusieurs Pollers pour des questions de répartition de charge (load-balancing).

Connexions avec les autres démons

L'Arbiter envoie la configuration du Poller sur le port 7771 de ce dernier.

Sa configuration correspond à la liste du ou des Schedulers du royaume auxquels il devra se connecter pour récupérer des checks à exécuter. Le Poller se connecte au(x) Scheduler(s) via le port 7768 de ce dernier.



Différents types de Pollers : Les Pollers TAG

L'architecture actuelle de Shinken Enterprise est très utile pour une organisation qui utilise le même type de Poller pour tous les checks. Mais il peut être nécessaire d'avoir différents types de Poller.

Cela peut être utile quand l'utilisateur a besoin d'avoir ses hôtes dans le même Scheduler (comme avec les dépendances) ou alors s'il a besoin d'avoir des hôtes ou des checks pris en charge par un Poller spécifique.

Ces vérifications peuvent être qualifiées à 3 niveaux :

- Hôte,
- Check,
- Commande.

La propriété "**poller_tag**" permet de qualifier une commande, un hôte ou un check.

La valeur de **poller_tag** est déterminée dans l'ordre de priorité suivant :

- On prend d'abord la valeur sur le check,
- puis sur l'hôte, si non défini sur le check,
- puis sur commande, sinon défini sur l'hôte.

Les Pollers peuvent être identifiés par plusieurs poller_tags.

S'ils ont des tags, ils ne prendront que les checks qui correspondent à ce tag.

C'est principalement utilisé dans un réseau en DMZ couplé avec l'option de Poller Passif (*le pourquoi dans le chapitre suivant*).

Poller Passif (exemple DMZ)

Cas particulier d'une DMZ

Il est possible d'avoir des checks à exécuter dans une DMZ (Zone Réseau Démilitarisé). Il est alors possible de définir un Poller en DMZ, et les Schedulers (et autres démons) dans le réseau interne, le LAN.

Un Poller dédié (Linux ou Windows) pourra être directement placé en DMZ pour des soucis de sécurité, de latence/temps d'exécution, ou, car l'environnement est différent (hors domaine AD, DMZ exclusivement Windows, etc..).

Dans ce cas, pour respecter la sécurité de l'espace démilitarisé, les connexions réseau de DMZ (Poller) vers LAN (Schedulers) ne sont pas autorisées par les Firewalls.

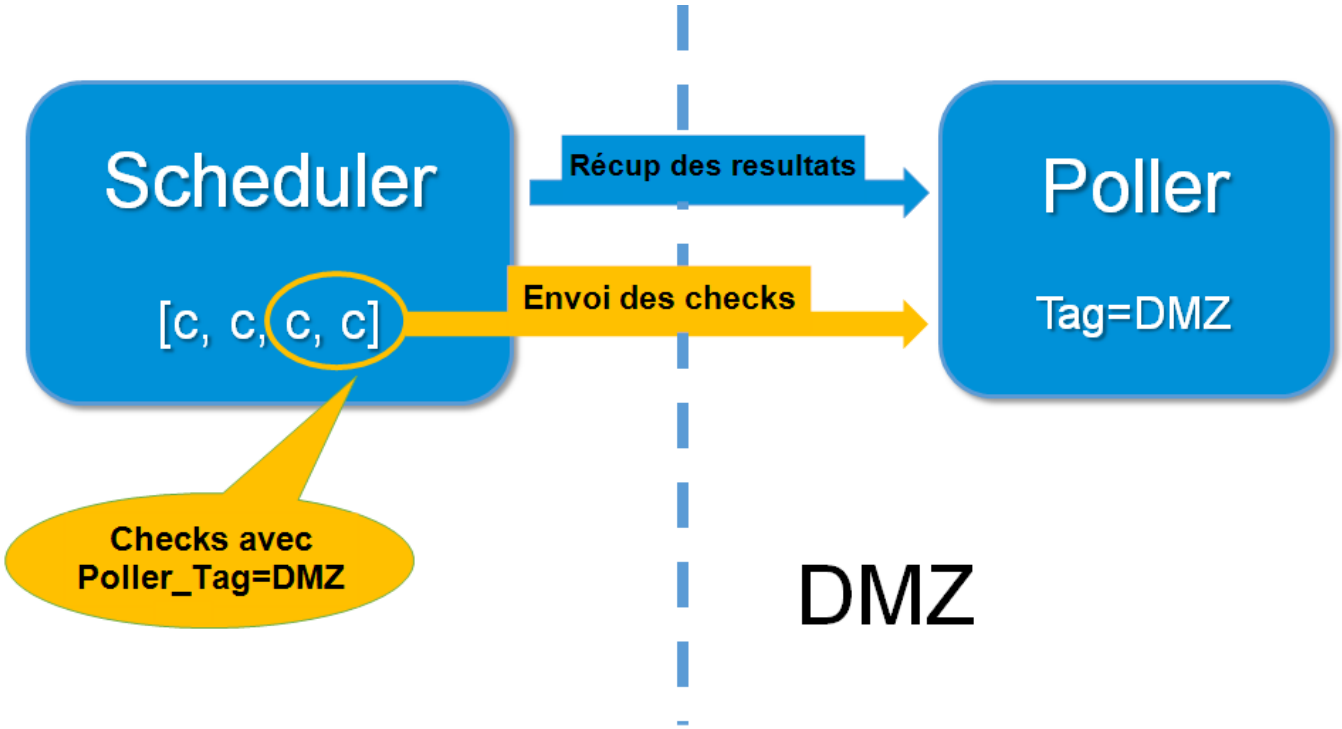
Pourtant, comme on l'a vu précédemment, par défaut, le Poller ouvre une connexion vers le ou les Schedulers.

Il est alors possible de définir le Poller comme **passif**. Le Scheduler va alors ouvrir une connexion vers le Poller.

Pour que le Scheduler n'envoie au Poller passif que les checks à réaliser dans la DMZ, il faut taguer les hôtes, les checks ou les commandes.

Seuls les hôtes, checks et commandes tagués seront alors lancés par ce Poller DMZ.

De cette manière, on peut toujours avoir des dépendances entre les hôtes dans la DMZ et le LAN, et pourtant être certain que les checks voulus sont effectués par le Poller dans la DMZ.



Notions de Scheduler "Rogue"

Lorsque des Pollers passifs sont ajoutés dans l'architecture Shinken, ce sont donc les Schedulers qui envoient directement les demandes d'exécutions de checks et qui récupèrent eux-mêmes les résultats auprès des Pollers.

Cependant, il se peut qu'un Scheduler communique à des Pollers alors qu'il ne devrait pas, par exemple :

- un Scheduler d'une infrastructure Shinken séparée, et mal paramétrée, qui continue à envoyer des requêtes à ce Poller alors qu'il ne devrait pas. On parle alors d'un Scheduler fantôme, ou encore d'un "Rogue" Scheduler.
- un Scheduler qui tente de parler à un Poller qui n'a pas encore reçu de configuration (temporaire si son Arbiter s'apprête à la lui envoyer)

Ces deux cas auront alors remonté dans le shinken-healthcheck (*voir la page Shinken-healthcheck - Vérifier le bon fonctionnement de Shinken Entreprise*) sous forme d'erreurs API, et ils remonteront également dans les logs, voici un exemple :

```
ERROR: [Shinken] A scheduler push me actions but it is not in my configuration so I drop these actions
ERROR: [Shinken] Maybe I'm not ready or there is a rogue scheduler
ERROR: [Shinken] Unknown Scheduler detected with host name:[master1] and sched_id[1]
```

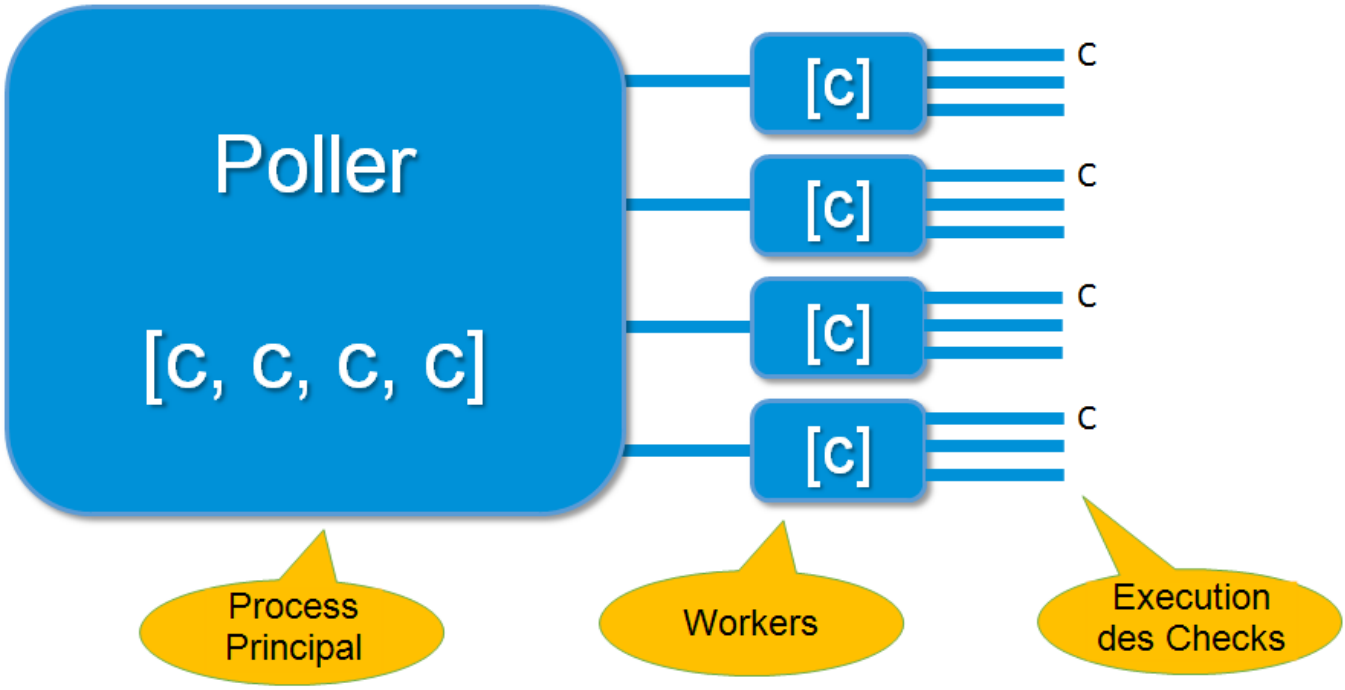
Dans les deux cas, l'information du "hostname" du serveur Scheduler permettra de prendre connaissance de ce dysfonctionnement.

Données

Que ce soit en mode actif ou passif, le Poller reçoit les checks (et donc les différentes lignes de commandes à exécuter) des Schedulers.

Il est important de noter que les sondes lancées par le Poller devront avoir un accès direct aux hôtes et devront récupérer des données venant de ceux-ci.

Fonctionnement du Poller



Résumé des connexions du poller

Mode Normal (actif)

Source	Destination	Port	Protocole	Note
Poller	Scheduler	7768	HTTP/HTTPS	

Le protocole est HTTP par défaut, et HTTPS si le paramètre SSL a été mis à 1.

Mode Poller Passif

Source	Destination	Port	Protocole	Note
Scheduler	**Poller**	7771	HTTP/HTTPS	

Le protocole est HTTP par défaut, et HTTPS si le paramètre SSL a été mis à 1.

Descriptions des variables

Propriété	Défaut	Description
poller_name	N/A	Cette variable est utilisée pour définir le nom du Poller.
address	N/A	Cette directive est utilisée pour définir l'adresse permettant de joindre ce Poller. Par défaut "localhost", changez-le par un nom DNS ou une adresse IP.
port	7771	Cette directive est utilisée pour définir le port TCP utilisé par ce démon.
use_ssl	0	Cette variable est utilisée pour définir si le Poller doit être contacté en HTTPS (*1*) ou HTTP (*0*). La valeur par défaut est *0* (HTTP).
keep_timeout_time	1200	Cette variable est utilisée pour fixer la durée durant laquelle le Poller va garder le timeout.
min_workers	0	Cette variable est utilisée pour définir le nombre de processus Worker utilisé par le Poller. Par défaut 0, correspond à 1 Worker par CPU.

processes_by_worker	256	Cette variable est utilisée pour définir le nombre maximal de commandes qu'un Worker est autorisé à exécuter en simultané.
polling_interval	1	Cette variable est utilisée pour définir le nombre de secondes à attendre avant que le Poller ne récupère les checks des Schedulers.
exec_stat_range	50, 100, 200, 300, 400, 1000, 5000, 15000	Cette variable est utilisée pour générer le tableau présent dans le check "Poller - Performance". Elle définit les différentes plages en millisecondes pour le temps d'exécution des checks.
display_statistics_compute_time_if_higher	100	Temps (en ms) de calcul à partir duquel le log affichant le temps de calcul des statistiques passe de DEBUG à INFO.
max_ram_percent	95	Cette variable permet de définir le taux de RAM physique occupée au-delà duquel il ne lancera pas de nouveau check.
max_cpu_queue_per_cpu	4	Cette variable permet de définir le nombre de processus par cœur dans la queue du processeur, au-delà duquel il ne lancera pas de nouveau check. Ce paramètre ne fonctionne pas sur des Pollers Windows.
timeout	3	Cette variable est utilisée pour définir le temps en secondes avant que l'Arbiter ne considère ce démon comme à l'arrêt. Si ce démon est joignable en HTTPS (use_ssl à 1) avec une latence élevée, il est conseillé alors d'augmenter cette valeur de timeout (l'Arbiter aura besoin de plus d'allers/retours pour le contacter).
data_timeout	120	Cette variable est utilisée pour définir le temps en secondes avant de considérer un transfert de configuration ou de données comme échoué.
max_check_attempts	3	Si le ping permettant de détecter la disponibilité réseau du nœud est en échec N fois ou plus, alors le nœud est considéré comme mort (par défaut, 3 tentatives).
check_interval	60	Intervalle de Ping toutes les N secondes.
satellitemap	N/A	Cette variable est utilisée dans le cas de royaume situé derrière un réseau NATé. Elle est de la forme d'une liste séparée par des "**,**" de valeur **nom-démon=address:port** Les démons ainsi listés seront contactés avec le couple address:port du paramètre au lieu de leur adresse dans leur .cfg. * Ceci permet ainsi à des démons derrière un réseau NAT d'échanger sur leur adresse locale au lieu de devoir ressortir sur leur adresse publique. *Exemple: daemon1=192.168.0.1:7768,daemon2=192.168.0.1:7771*
modules	N/A	Cette variable est utilisée pour définir tous les modules que le Poller va charger.
realm	N/A	Cette variable est utilisée pour définir le royaume où sera mis le Poller. Si aucun n'est spécifié, il sera rattaché au royaume par défaut.
manage_sub_realms	1	Cette variable est utilisée pour définir si le Poller acceptera les tâches du Scheduler provenant des sous-royaumes de son royaume.
poller_tags	None	Cette variable est utilisée pour définir le Poller tag. Par défaut None, qui représente les éléments non tagués.
passive	0	Cette variable est utilisée pour définir le mode de connexion. Par défaut 0, le Poller se connecte au Scheduler pour récupérer ou retourner les informations. Si 1, le Scheduler se connecte au Poller pour pousser ou récupérer les informations.
spare	0	Cette variable est utilisée pour définir si le Poller peut être géré comme spare (ne chargera la configuration que si le maître tombe).
enabled	1	Cette variable est utilisée pour définir si le Poller est activé ou non.
`vmware__statistics_compute_enable`	1	Cette variable permet d'activer ou de désactiver la collecte des statistiques de VMware.

Exemple de définition

Dans le répertoire **/etc/shinken/pollers/**, voici un exemple de définition qui permet la définition d'un Poller (à placer dans un fichier CFG) :

⚠️ Il est conseillé d'éditer les fichiers .cfg avec l'encodage utf-8

```
#=================================================================================
# POLLER
#=================================================================================
# Description: The poller is responsible for:
# - Execute checks
#=================================================================================
```

```
define poller {

# Shinken Enterprise. Lines added by import core. Do not remove it, it's used by Shinken Enterprise to
update your objects if you re-import them.
    _SE_UUID            core-poller-d571cb8c5add11e5846f080027f08538
    _SE_UUID_HASH       e04b7dc95665955df040e3e2e6767b29
# End of Shinken Enterprise part


    #======== Daemon name and address =========
    # Daemon name. Must be unique
    poller_name         poller-master

    # IP/fqdn of this daemon (note: you MUST change it by the real ip/fqdn of this server)
    address             localhost

    # Port (HTTP/HTTPS) exposed by this daemon
    port                7771

    # 0 = use HTTP, 1 = use HTTPS
    use_ssl                    0

    # Duration in which we will keep the timeout.
    keep_timeout_time   1200


    #======== Daemon performance sizing =========
    # Number of worker process to launch. 0 = 1 per CPU
    min_workers               0

    # Number of maximum commands a worker is allowed to launch in the same time
    processes_by_worker       256

    # Specify the number of seconds the Poller should wait before retrieving jobs from the Scheduler
    polling_interval          1

    # Ranges for the check : poller statistics
    #exec_stat_range    50, 100, 200, 300, 400, 1000, 5000, 15000

    # Time (in ms): if the time to compute stats (for shinken checks) is higher
    #               than this time, then the log will be in INFO level
    # Default: 100  (ms)
    # display_statistics_compute_time_if_higher    100

    # Percentage of used physical RAM beyond which the poller will not launch any new check
    max_ram_percent           95

    # Number of maximum runnings processes in the CPU Queue per CPU. Default value is 4.
    #max_cpu_queue_per_cpu    4

    #======== Daemon connection timeout and down state limit =========
    # timeout: how many seconds to consider a node don't answer
    timeout                   3

    # data_timeout: how many second to consider a configuration transfert to be failed
    # because the network brandwith is too small.
    data_timeout              120

    # max_check_attempts: how many fail check to consider this daemon as DEAD
    max_check_attempts        3

    # Check this daemon every X seconds
    check_interval            60



    #======== Modules to enable for this daemon =========
    # Available:  None for this daemon
    modules
```

```
    #======== Realm and architecture settings =========
    # Realm to set this daemon into
    realm                   All

    # manage_sub_realms 1 = (default) take data from the daemon realm and its sub realms
    #                   0 = take data only from the daemon realm
    manage_sub_realms       1

    # In NATted environments, you declare each satellite ip[:port] as seen by
    # *this* daemon (if port not set, the port declared by satellite itself
    # is used)
    #satellitemap    scheduler-1=1.2.3.4:7768, scheduler-2=1.2.3.5:7771

    # Poller tags are managed tags. The hosts, checks and commands can have specific tags, and
    # this poller will only take them if the element tag is list in the poller
    # Reserved tag name:  None  =  untaggued element (default value)
    #poller_tags            None

    # passive:  0 = poller will connect to the scheduler to take/returs jobs
    #           1 = scheduler will connect to this poller to push/take back jobs
    passive                 0

    # spare: 0 = this poller is always active
    #        1 = this poller will be activated by the arbiter is another poller is dead
    spare                   0


    #======== VMWare / ESXi ==========
    # 1 (default) = if vmware get the ESXi CPU stats value, 0 = do not get value
    vmware__statistics_compute_enable     1


    #======== Enable or not this daemon =========
    # 1 = is enabled, 0 = is disabled
    enabled                 1

}
```

Paramétrage spécial dans le cas de nombreux pollers et nombreux schedulers sur de gros environnements

Description du bug de surcharge

Un bug connu en cours de résolution demande un paramétrage spécial des Pollers dans le cas où il y a plusieurs Pollers et plusieurs Schedulers (*non spares*) dans un même royaume.

Le Poller est fait pour se limiter sur sa charge, mais quand il y a plusieurs Schedulers, dans la version actuelle, il fait la demande pour cette disponibilité de CPU (*disons ici 4s de temps CPU pour une machine avec 4CPU qui n'est pas chargée actuellement*) mais à tous les Schedulers à la fois.

Si ces derniers ont beaucoup de sondes à lancer, alors le Poller va recevoir non pas 4s de travail, mais **4s*nombres de Schedulers**.
Ceci peut le surcharger et provoquer des dysfonctionnements en cascades:

- les sondes prennent trop de temps, elles vont tomber en erreurs de timeout
- voir ne pas avoir le temps de revenir au Scheduler avant que ce dernier ne décide de donner la sonde à un nouveau Poller (*et donc avoir un effet domino, ce nouveau Poller tombant aussi en surcharge*)

Paramétrage pour éviter le bug avant sa résolution

Pour éviter cela, il faut descendre le paramètre permettant au Poller de se protéger et ne pas prendre de nouvelles sondes avant que son load average (*à 1min*) ne redescendre à un niveau acceptable.

Le paramétrage conseillé est alors dans ce cas:

- **processes_by_worker** à **2**

    - ceci permet de limiter le nombre maximum de sondes qui s'exécutent en parallèle par worker (*sachant que par défaut on a un worker par CPU sur le serveur*)
- **max_cpu_queue_per_cpu** à **2**
    - ceci permet de définir le niveau acceptable de surcharge.
    - Ainsi, le Poller ne prendra pas de nouvelles sondes avant que son load average (*à 1min*) ne redescendre à **2 * nombres de CPUs**.

IMPORTANT

Ce réglage peut-être contre productif si les sondes ont beaucoup de latence réseau (le processus attend la réponse de l'équipement en face), car pendant ce temps il est en pause et comme on lance moins de processus, les CPUs sont inutilisés.