Mémoire de masse

Structure des disques

Ordonnancement de disques

RAID

Structure des SSD

Structure d'un disque

Structure tridimensionnelle: plateaux, secteurs, cylindres

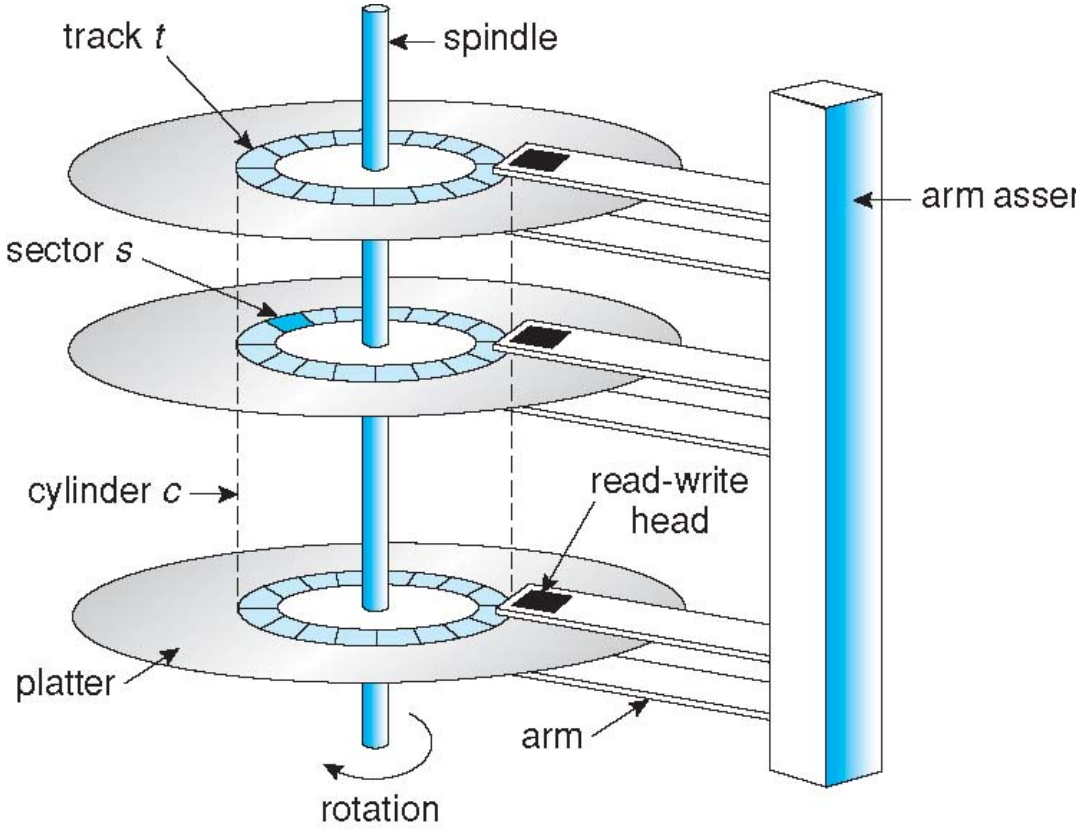
Piste = plateau ∩ cylindre

Bloc = piste ∩ secteur

Têtes *volent* sur les plateaux

De 3600tpm à 15000tpm

Têtes se déplacent radialement



seek time: temps de déplacement de la tête

rotational latency: temps de déplacement du bloc par rotation

Performance d'un disque

Capacité: de l'ordre de 2TB en 2016, en blocs de 4kB

rotational latency moyenne:

temps pour $\frac{1}{2}$ tour

$3600tpm$ $8.3ms$

$7200tpm$ $4.2ms$

$15000tpm$ $2ms$

seek time moyen:

temps pour $\sim\frac{1}{3}$ du rayon

$3ms - 12ms$ moyen

$\sim\frac{1}{2}ms$ piste à piste

Changement de tête: $\sim 1ms$

Bande passante (*bandwidth*): $\sim 100MB/s$

Vue externe d'un disque

Les détails physiques sont cachés

- Le SE ne voit qu'un tableau de N blocs logiques de taille fixe
- Reçoit des requêtes *read* et *write* sur ces blocs
- Accès séquentiels censés obtenir meilleure performance

Notion de *secteur* peut-être invalide

Blocs endommagés cachés par *remapping*

Le disque inclus une mémoire cache

Maintient une queue de requêtes en cours (*command queuing*)

SAN, NAS, Storage arrays

Storage array

- contrôleur connecté à plusieurs/beaucoup disques
- Vu de l'extérieur comme un *autre* ensemble de disques

Storage area network (SAN)

- Réseau spécialisé pour connecter des disques et des machines
- Disques *dédiés*. Facilité d'ajouter ou enlever disques

Network attached storage (NAS)

- Disques vus soit comme des disques ou des systèmes de fichiers
- Réseau standard. Disques peuvent être *partagés*.

Ordonnancement de disques

Temps d'accès fortement influencé par les déplacements de têtes

L'ordre des accès influence fortement le temps total

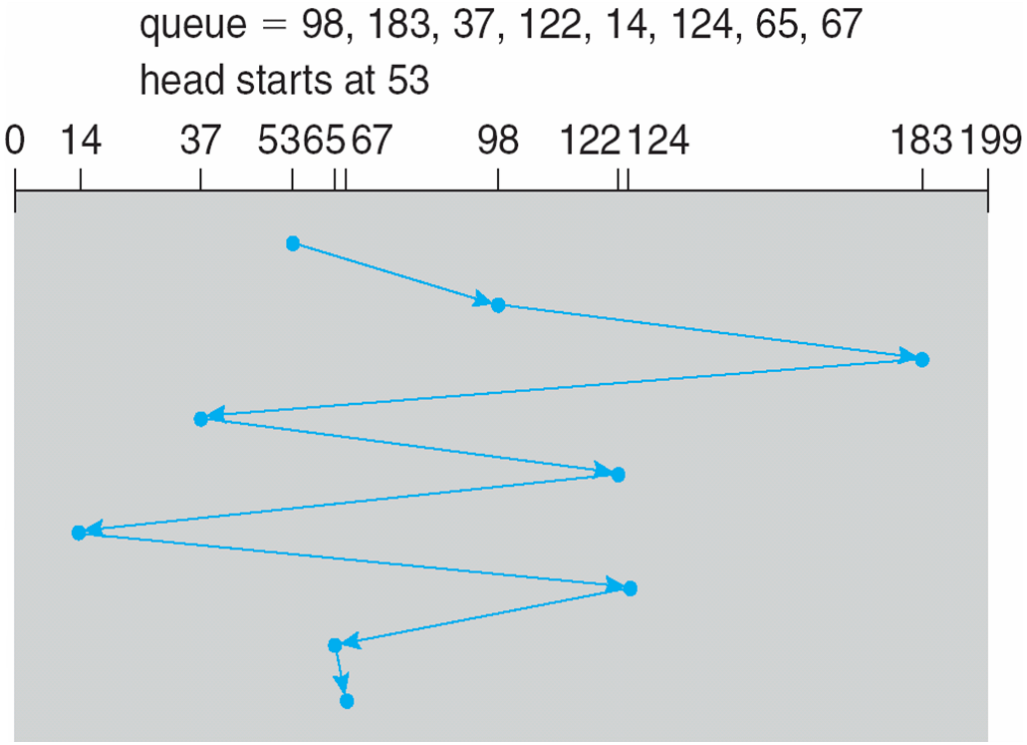
Une queue de requêtes par disque (dans le SE et/ou le disque)

Si la queue est vide: pas de différence

Sinon, choix d'algorithmes d'ordonnancement

First Come First Serve (FCFS/FIFO)

Garde l'ordre d'arrivée des requêtes

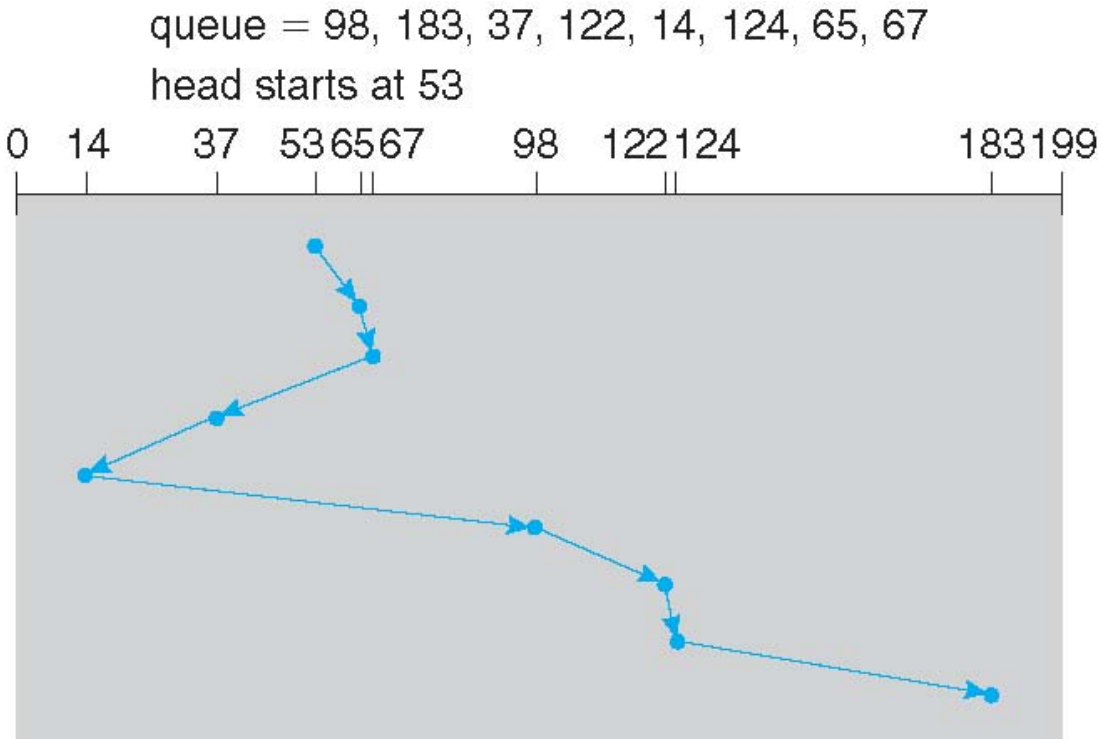


Déplacement total sur cet exemple: 640 unités

Shortest Seek Time First (SSTF)

Choisi toujours la requête la plus proche de la position actuelle

Similaire à Shortest Job First: peut souffrir de famine

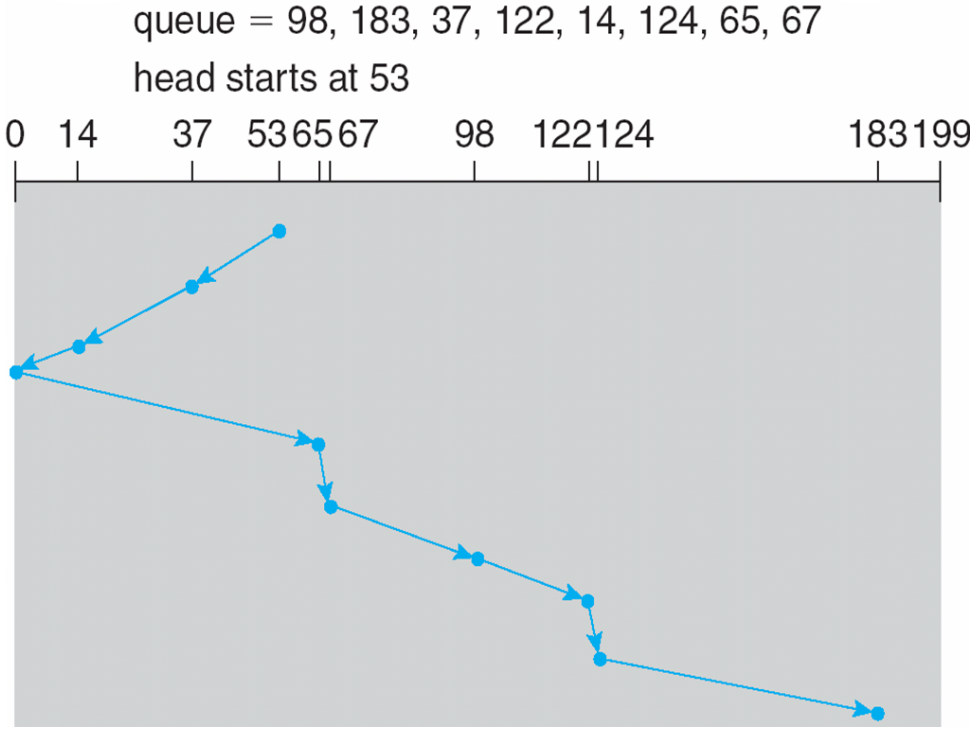


Déplacement total sur cet exemple: 236 unités

SCAN

La tête parcourt toute la surface dans un sens puis dans l'autre

Aussi appelé algorithme de l'ascenceur

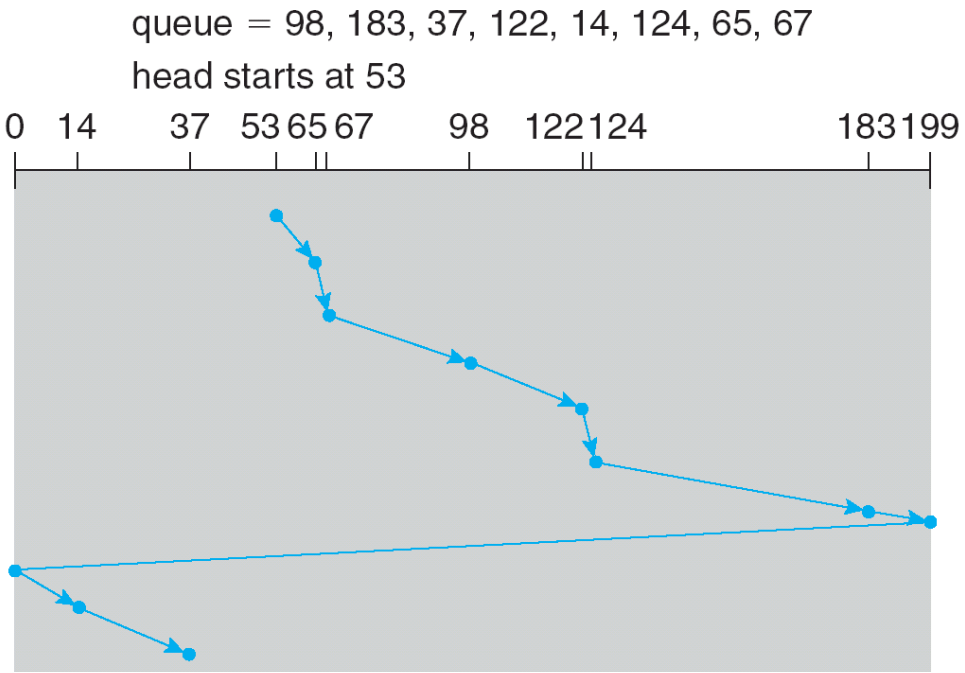


Déplacement total sur cet exemple: 208 unités

SCAN circulaire (C-SCAN)

La tête parcourt la surface toujours dans le même sens

Diminue l'attente maximum par rapport à SCAN

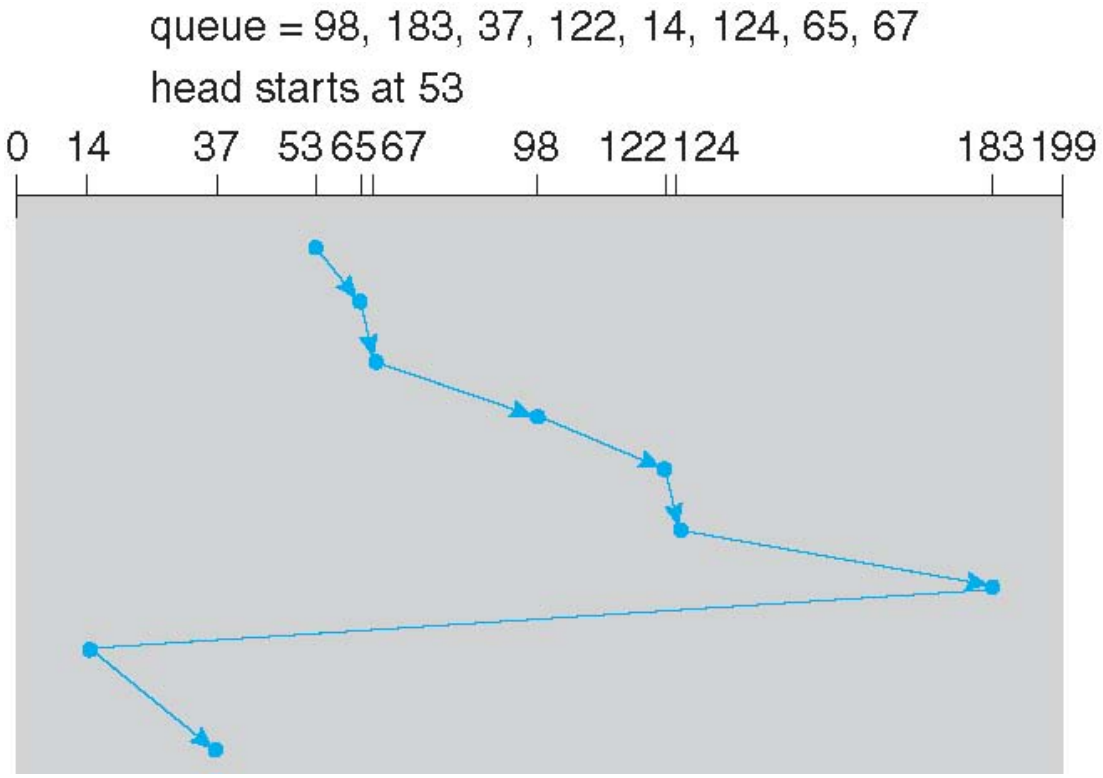


Déplacement total plus grand, mais: 1x200 < 2x100

LOOK circulaire (C-LOOK)

LOOK = SCAN sans aller vraiment jusqu'au bout

C-LOOK = C-SCAN en évitant aussi les déplacements inutiles



Fonctionnement de l'ordonnanceur

Le SE ne peut pas vraiment tenir compte de la *latence*

Le disque a son propre ordonnanceur

Nombre de requêtes limité dans la queue du disque

Dépendances entre requêtes

- L'ordre des écritures visible en cas de panne
- L'ordonnancement doit en tenir compte

Gestion des disques

Un “disque” est une notion abstraite: un tableau de blocs consécutifs

Un disque est divisé en *partitions*

On peut aussi joindre deux disques en un grand disque virtuel

On peut voir les partitions comme des disques

RAID: Redundant Array of Inexpensive Disks

Très difficile d'accélérer les disques ou augmenter leur fiabilité

Utiliser plusieurs disques à la place

Copies redondantes sur plusieurs disques, pour la fiabilité

Accès parallèles à plusieurs disques, pour la performance

Plusieurs structures possibles, selon les besoins

RAID-0: Striping

Combiner N disques PD_i en un grand disque LD

Données réparties finement sur tous les disques

Divisé en *stripes*. Stripe S placée sur disque $S \bmod N$

$$\textit{taille}(LD) = \sum \textit{taille}(PD_i)$$

$$\textit{bande\_passante}(LD) \simeq \sum \textit{brande\_passante}(PD_i)$$

$$\textit{IOPS}(LD) \simeq \sum \textit{IOPS}(PD_i)$$

Mais:

$$\textit{latence}(LD) \simeq \textit{latence}(PD_i)$$

$$\textit{fiabilité}(LD) \simeq \frac{1}{N} \times \textit{fiabilité}(PD_i)$$

RAID-1: Mirroring

Combiner N disques PD_i en un disque LD de même taille

Données copiées N fois: chaque disque est une copie des autres

$$\textit{fiabilité}(LD) \simeq N \times \textit{fiabilité}(PD_i)$$

$$\textit{bande\_passante\_RD}(LD) \simeq \sum \textit{brande\_passante\_RD}(PD_i)$$

$$\textit{IOPS\_RD}(LD) \simeq \sum \textit{IOPS\_RD}(PD_i)$$

$$\textit{latence}(LD) < \textit{latence}(PD_i)$$

Mais:

$$\textit{bande\_passante\_WR}(LD) < \textit{brande\_passante\_WR}(PD_i)$$

$$\textit{IOPS\_WR}(LD) \simeq \textit{IOPS\_WR}(PD_i)$$

$$\textit{taille}(LD) \simeq \textit{taille}(PD_i)$$

RAID-4: Parity

Striping sur N disques plus un disque supplémentaire de *parité*

En lecture: comme RAID-0 avec le disque de parité inutilisé

Fiabilité bien meilleure que RAID-0: un disque peut mourir sans perte

Degraded mode: fonctionnement avec un disque en moins

Mais, en écriture: pire que RAID-1

Chaque écriture touche au disque de parité

$$\textit{bande\_passante\_WR}(LD) < \textit{bande\_passante\_WR}(PD_i)$$

Le calcul de la parité peut nécessiter des lectures supplémentaires

$$\textit{IOPS\_WR}(LD) \simeq \frac{1}{2} \times \textit{IOPS\_WR}(PD_i)$$

RAID-5: Distributed parity

Comme RAID-4, mais avec stripes de parité réparties sur les disques

En lecture: $\frac{N+1}{N}$ fois mieux que RAID-4 (tous les disques participent)

Fiabilité identique à RAID-5: un disque peut mourir sans perte

Degraded mode: un peu plus complexe que RAID-4

En écriture: beaucoup mieux que RAID-4

Le coût de la parité réparti entre tous les disques

$$\textit{bande\_passante\_WR}(LD) \simeq \frac{N}{2} \times \textit{bande\_passante\_WR}(PD_i)$$

Le calcul de la parité peut nécessiter des lectures supplémentaires

$$\textit{IOPS\_WR}(LD) \simeq \frac{N}{4} \times \textit{IOPS\_WR}(PD_i)$$

Plus sur les RAIDs

RAID-6: comme RAID-5 mais avec un deuxième bit de “parité”

Hot spare: disque inutilisé prêt à remplacer un autre

RAID-nm: un RAID-m constitués de disques logiques de type RAID-n

RAID-01: un mirroir de deux RAID-0 (striping) identiques

RAID-10: un striping de deux RAID-1 (mirroring)

RAID-10 donne la meilleure performance après RAID-0

Solid State Disks (SSD)

Disque constitué de mémoire non-volatile à semi-conducteur

De nos jours: NAND-Flash ou NOR-Flash

NOR: performance et longévité; NAND: densité et coût

Une USB-key est un SSD qui vise le meilleur prix par MB

Un vrai SSD vise une meilleure performance par $

Performance beaucoup plus élevée qu'un HDD:

- Bande passante: ~500MB/s
- Latence: $<$0.1ms

Meilleure fiabilité mécanique; basse consommation

NAND-Flash

Mémoire divisée en *erasure blocs* (e.g. 128kB)

erasure blocs subdivisé en *pages* (e.g. 8kB)

Opérations:

- `read_page`: pas de surprise
- `write_page`: ne peut que changer des bits à 0
- `erase_bloc`: met tous les bits à 1

Phénomène d'usure: un bloc survit à ~10'000 `erase_bloc`

Caractéristiques très variables

Flash-Transation-Layer (FTL)

SSDs habituellement cachent leur NAND derrière une couche magique

- Expose un tableau de N blocs logiques de taille fixe (e.g. 4kB)
- Avec seulement `read_bloc` et `write_bloc`

Performance en écriture pas toujours prévisible:

- Un `write_bloc` peut devoir lire+effacer+réécrire un *erasure bloc*
- *Wear-leveling*: Le FTL doit répartir les *erase_bloc* uniformément
- En fait, `write_bloc` va généralement écrire *ailleurs*!

Table auxiliaire garde position physique d'un bloc logique

Blocs devenus inutilisés, récupérés par une sorte de GC

Exemple de FTL

Chaque *bloc logique* correspond à une *page*

Chaque *erasure bloc* contient un descripteur de ses *bloc logiques*

En mémoire:

- Une table de traduction *bloc logique* $\Rightarrow$ *page*
- Une table des pages libres

`read_bloc` simple et efficace:

1. Utilise la table pour trouver la page
2. Lis la page

Au démarrage: lire tous les descripteurs pour reconstruire les tables

Exemple de FTL: écriture

Un *erasure bloc courant* en cours d'écriture

Lors d'un *write_bloc*

- Écrit dans la prochaine *page* libre du *erasure bloc courant*
- Met à jour les tables des blocs et des pages libres

Quand *erasure bloc courant* est plein:

- Écrire le *descripteur*
- Choisir un autre *erasure bloc*
- Lire ses pages non-vides; `erase_bloc`; Réécrire ses pages