Par :: Yann Guidon :: whygee@f-cpu.org :: http://ygdes.com ::

Installation, fiabilisation et entretien des disques durs

Il existe de nombreuses méthodes et précautions, certaines spécifiques à GNU/Linux, pour éviter les pertes accidentelles de données, les réparer, circonscrire leur importance et peut-être même les récupérer. Les techniques préventives sont aussi bien méthodologiques (sauvegardes consciencieuses) que techniques. Parmi ces dernières, la préparation du support, couramment un disque dur ou une mémoire Flash, est primordiale, que le support soit neuf ou d'occasion !

Acte I : Pas de précipitation

Un souci omniprésent de fiabilité

Les mémoires amovibles comme les clés USB qui sont à la mode, mais encore chères, supportent un nombre limité d'écritures et leurs capacités sont encore relativement modestes. Par contre, les disques durs sont déjà partout.

Les capacités comme les performances augmentent alors que leur prix baissent. Par rapport aux semi-conducteurs (micro-processeurs et mémoires), le prix des disques est moins sensible aux variations cycliques du marché.

Pour l'instant, un Gigaoctet coûte environ 50 centimes d'euro et cela va encore baisser. Le temps d'accès est plus court qu'un CD-ROM ou un DVD-ROM à un prix concurrentiel et sans les contraintes de retrouver la bonne galette dans une pile, puisque tout est dans une petite boite pouvant contenir l'équivalent d'une centaine de DVD. Les disques durs servent donc de plus en plus à l'archivage, ce qui augmente l'importance de leur pérennité. Ce sont des concentrés technologiques extrêmement complexes et leur pièces, mécaniques ou électroniques, ne sont pas infaillibles.

L'augmentation des volumes de production entraîne de surcroît des compromis entre fiabilité et coût, sur lequel les termes de la garantie sont calculés par chaque constructeur. Contrairement aux semi-conducteurs, l'usure mécanique naturelle agit pour limiter la durée de vie.

Ne vous attendez donc pas à léguer votre sauvegarde à vos petits-enfants. De toute façon, la capacité des supports aura encore explosé d'ici là.

Et dans ce monde bien complexe, l'utilisateur doit pouvoir faire des choix répondant à ses besoins, surtout quand il s'agit de faire un investissement fiable. L'objectif est de faire durer les données le plus longtemps possible à moindre coût.

Analyse du marché

Une question bien sensible : quel modèle choisir, quel fabricant favoriser ?

Chaque fabricant a connu ses histoires d'horreur et des séries ratées. En ce qui me concerne, je viens de perdre trois disques durs de la même marque, coup sur coup, dont deux disques d'un même lot. Cela fait réfléchir.

Cet article devait donc s'appeler « les malheurs de Whygee », mais je ne pense pas être le seul à disposer d'une pile de disques durs abîmés ou inutilisables, sans parler de ceux qui sont retournés au magasin lorsque la garantie était encore valable. Et si vous n'êtes pas électronicien(ne), cela ne vous intéresse pas de dessouder les composants qui les constituent.

Cela vaut d'ailleurs pour toutes les pièces d'un ordinateur. Cependant, elles ne font que transmettre ou convertir les données et leur fiabilité est moins critique : « griller » une carte vidéo ou un processeur ne réduit pas forcément vos travaux à néant. Les disques durs sont donc des périphériques à part.

Les premières recommandations sont simples : d'abord se renseigner à propos des modèles disponibles, grâce à Google par exemple. Ce qui est aujourd'hui tellement facile que personne ne songe à le faire (avant qu'il ne soit trop tard).

Cependant, ce n'est payant que pour les séries présentant des défauts majeurs et évidents. Les soucis de fiabilité des disques durs n'apparaissent parfois que lorsqu'une technologie est largement distribuée. Le temps que l'information fasse le tour du Net, les magasins seront déjà remplis avec d'autres modèles débugués et encore plus compétitifs.

Ensuite, essayer simultanément (si cela est possible) plusieurs marques. Ainsi les dégâts seront réduits si un des disques appartenait par malchance à une « série maudite ».

De plus, certains disques, plus chers, ont des composants de meilleure qualité et des caractéristiques (MTBF, conditions et environnement de fonctionnement) plus étendues.

Par exemple, les disques durs pour serveurs sont conçus pour travailler

:: Illus. 1 :: Vous avez peut-être aussi des disques durs en mauvais état.
Combien renferment des données importantes ? Certains pourraient-ils être réutilisés ? ::

24h sur 24 alors que les modèles « économiques » ou grand public sont conçus pour être arrêtés régulièrement. Les contraintes thermiques (les cycles dilatation/rétractation) inhérentes sont différentes, ce qui se reflète évidemment dans le prix.

Un autre facteur est la capacité : les plus petits sont moins chers, donc avec un volume de production plus grand pour compenser la marge réduite. Ils deviennent obsolètes plus vite car les disques plus gros et plus récents vont se vendre moins vite, dans un premier temps, en attendant que la production de ce modèle s'accélère, que le rapport prix/capacité soit plus intéressant que celui des modèles moins chers et que les réserves de ces derniers s'épuisent.

C'est un choix à double tranchant : les gros disques ont un meilleur rapport capacité/prix et corrigent les problèmes rencontrés sur les modèles précédents. Cependant, ils sont plus chers et les éventuels défauts sont encore inconnus.

Comme souvent, le compromis économique est difficile. Faut-il acheter un disque plus cher (mettre les oeufs dans un même panier probablement mieux rembourré) ou plusieurs disques économiques (plusieurs paniers pour réduire les dégâts) ?

Il ne faut pas non plus oublier le « prix » à l'utilisation, en particulier le bruit ou les vibrations qu'un ou plusieurs disques génèrent ainsi que la consommation électrique qui n'est pas négligeable. Elle est proportionnelle à la chaleur générée et est donc à dissiper.

Le cérémonial de l'achat

Avec un peu d'astuce, il est parfois possible de tester un disque dur neuf dans le magasin où il est acheté. Mais de plus en plus cela devient complètement utopique en raison de la taille des disques actuels et du temps nécessaire à les scanner complètement, comme nous verrons plus loin.

La recommandation est, si l'opportunité s'offre à vous, de choisir un magasin spécialisé qui acceptera la vérification du matériel sur place. Certains font bien le montage de PC et la vérification de la carte mère ou de cartes d'extension, gratuitement qui plus est.

Cela est possible dans les petits magasins, comme ceux de la rue Montgallet à Paris. Aucun espoir dans une grande surface, évidemment.

L'intérêt de cette démarche est double : la certitude pour le vendeur et le client de la compatibilité et de la fonctionnalité du produit, mais en plus, en cas d'erreur ou de défaut évident, on économise des allers-retours au magasin.

La démarche de vérification lors de l'achat prend une autre mesure quand il s'agit d'accessoires pour ordinateur portable : le système entier peut être apporté et testé en conditions réelles.

Par exemple, un périphérique PCMCIA ou USB peut être essayé et les performances ou la compatibilité avec le système d'exploitation sont sans équivoque. Avec un adaptateur IDE vers USB2, vous pourrez même tester un disque dur vous-même.

Il est raisonnable de tester les quelques premiers Gigaoctets en magasin. Cela met le disque en condition d'utilisation et révèle aussi d'autres défauts éventuels au niveau des circuits de contrôle.

De plus, si le disque commence sur des secteurs valides, la table des partitions ne sera pas corrompue par des défauts, ce qui est le minimum vital pour installer un système de fichiers.

Apportez aussi votre ping^Wmanchot préféré !

Accessoire supplémentaire : prévoir un CDROM *bootable* avec une distribution « live ». Avec un peu de chance, un vendeur suffisamment disponible (il ne faut pas non plus empêcher les autres clients d'être servis) aura un ordinateur de test sous la main et bootera avec le CDROM pour tester le disque sur un vrai port ATA rapide.

Pourquoi un *LiveCD* ? Libre à vous de faire confiance à MSWindows, certainement installé dans l'ordinateur du magasin. Un GNU/Linux bootable (vous avez le choix de la distribution) procure en tout cas une visibilité inégalable dans les couches de bas niveau (`dmesg` est notre ami) et peut renseigner sur des caractéristiques qui ne sont pas mentionnées sur l'étiquette du produit (`hdparm` est aussi notre ami). Et ce ne sont que les tâches les plus simples et les plus évidentes.

Un LiveCD permet ainsi un contrôle précis de nombreux paramètres à tester, contrairement à des utilitaires intégrés à d'autres systèmes d'exploitation destinés au grand public.

... et plus si affinité

Mais ce n'est parfois pas suffisant et le kernel du LiveCD peut contenir un bug lié au *chipset* ou que sais-je encore (comment ça, « ça sent le vécu » ?).

Le magasin utilise peut-être un ordinateur fait de bric et de broc, avec une carte mère toute récente et utilisant des composants peu orthodoxes. Donc un échec de vérification ne veut pas toujours dire que le disque est défectueux.

Pour garantir que le problème vient effectivement du disque, il faut lancer l'utilitaire de diagnostic du constructeur (souvent disponible sur son site web). De toute façon, en cas de retour au service après-vente, c'est ce même utilitaire

:: Illus. 2 :: Avec cette paire de gros disques, je devrais être à l'abri u besoin et des accidents si je prends toutes les précautions :-) ::

qui sera lancé par le technicien avant même de considérer un échange ou un remplacement.

Il est possible d'apporter un autre CD bootable comme The Ultimate Boot CD (voir plus loin dans l'article). Celui-ci contient sur un seul CDROM les utilitaires de la plupart des constructeurs, ce qui laisse le choix de la marque et du modèle en fonction de leur disponibilité dans le magasin.

Et pour remercier le vendeur de sa patience, vous pouvez lui laisser le CDROM à la fin, ce qui l'aidera peut-être dans son travail (ou vous aidera à négocier une ristourne).

Une fois de retour à la maison, lancez tout de suite la vérification complète, totale, exhaustive de la surface.

Plus vous attendrez, moins vous aurez de chances de faire remplacer le disque (s'il se révèle défectueux) par le magasin. Au-delà d'une semaine après l'achat, les démarches sont généralement plus pénibles.

On ne sait vraiment jamais

Pour les \/\/@R1ØrD5, il existe encore une précaution supplémentaire. Cela consiste, si le budget le permet, à acheter deux disques (ou plus) strictement identiques.

La première application logique est d'utiliser les disques dans un système RAID, décrit par d'autres articles. A défaut de RAID, monter les deux disques comme maîtres de leur canal IDE respectif est la meilleure chose à faire. Bande passante accrue, haute disponibilité, faites le choix de configuration qui convient à vos besoins et vos compétences.

Si la carte mère ne supporte pas le RAID matériel, il existe des cartes PCI économiques remplissant cette fonction. Cela libère des emplacements IDE sur la carte mère, qui utilise peut-être déjà d'autres disques ou lecteurs. Sinon, Linux gère le RAID logiciel.

Achetés le même jour dans le même carton, les disques d'un même modèle auront (normalement) des numéros de série consécutifs (à vérifier !). Donc des composants identiques, éventuellement interchangeables.

The swaptrick

Si jamais un des disques casse pour une raison ou une autre, et si cette casse ne concerne pas la chambre hermétique, il sera alors peut-être possible de récupérer les données du disque en montant la carte contrôleur du disque indemne.

Le risque de perdre les données de l'autre disque (en cas d'erreur) n'est pas négligeable.

Il faut aussi bien connaître l'électronique pour éviter les mauvaises manipulations. Les connecteurs peuvent être très délicats ou nécessiter du matériel spécial pour les enlever. Cela ne s'improvise pas.

Cette technique part aussi du principe que seule la platine de contrôle extérieure est endommagée.

En cas de choc électrique (électricité statique, foudre, inversion de polarité...), l'électronique à l'intérieur de la zone hermétique (qui est extrêmement sensible) a de fortes chances d'être aussi endommagée.

A ce niveau, il est impossible d'intervenir avec succès sans des moyens irraisonnables.

De plus, cette greffe implique que la garantie des deux disques parte en fumée.

A vous de voir si les données valent la perte de deux unités en cas de complications.

Mise en perspective

De toute façon, la garantie se résume souvent en pratique au remplacement de l'unité défectueuse. Les données ne sont même pas récupérées.

En passant, il faut préciser que les services de récupération de données sur des disques endommagés sont absolument hors de prix, sauf pour une grosse entreprise.

Mais y avoir recours voudrait dire que sa politique de conservation des données critiques est très mauvaise.

A côté de cela, tenter une récupération par un échange de la carte contrôleur coûte énormément moins cher et l'ubiquité des unités permet éventuellement de garder un double des données.

En fait, la duplication est la méthode la plus fiable pour garantir la disponibilité et la conservation des données à coût réduit, surtout à une époque où une unité neuve de premier prix coûte environ 50 euros.

Dans le cas où les données ne sont pas dupliquées (pour doubler l'espace de

stockage par exemple), avoir un double de l'unité permet encore de tenter une greffe temporaire durant l'extraction des données. Enfin, il faut être conscient que la durée de garantie de tout appareil est typiquement calculée en fonction de la durée de vie estimée par le constructeur, pas en fonction de son utilisation réelle. Comme par hasard, les incidents arrivent souvent juste après l'expiration de la garantie. La décision d'échanger le contrôleur peut alors être tristement simple à prendre, encore faut-il avoir la doublure du disque...

L'intégrité zéro n'existe pas.

Le disque installé dans l'ordinateur, vous êtes sur le point d'installer les données et les logiciels qui manquaient de place. STOP !

Vous vous sentez peut-être chanceux, tous vos disques antérieurs vous ont probablement donné satisfaction et les composants de votre ordinateur sont d'excellente qualité. Alors dites-vous que tout cela est révolu. Songez d'abord à ceci : un disque dur actuel contient environ mille milliards de bits. A cette échelle, la question n'est pas de savoir s'il y a des bits défectueux, mais combien.

Il est impossible de répondre avec précision : les disques durs modernes sont incroyablement sophistiqués et les techniques utilisées pour coder les bits avec une telle densité sont le résultat de décennies de recherches acharnées. On sait par contre, à notre niveau d'utilisateur, que des zones d'un disque ou d'une mémoire Flash sont réservées pour remplacer les secteurs défectueux.

Le contrôleur intégré effectue la détection des erreurs et la substitution sans que l'utilisateur ou même le système d'exploitation ne s'en aperçoivent, à moins de passer par une interface spéciale, si elle existe. Après l'assemblage des pièces, les tests en usine vont analyser la surface du disque et détecter, marquer et remplacer les secteurs défectueux, lors du formatage de bas niveau. Mais entre la fabrication et la vente, des mois se sont écoulés et les différentes conditions de stockage et de transport (poussières, humidité, températures, vibrations, pressions) ont peut-être commencé à faire travailler l'ensemble. Ou alors (imaginons le pire) le magasin pratique la « remballe » des pièces retournées et remboursées.

L'astuce du chef

Pour les adeptes des paires de disques durs, il existe un moyen simple et efficace de réduire le temps du balayage de la surface : il suffit de brancher chaque disque sur un canal IDE individuel et de lancer badblocks -w dans deux terminaux différents.

En utilisant des câbles différents, chaque disque peut travailler à sa vitesse maximale et une carte-mère décente gère ces deux flux sans problème. Le temps d'exécution de badblocks -w ne change donc pas, mais pouvoir le lancer sur deux disques différents divise le temps total de test par deux !

Le doute est encore plus évident dans le cas de l'acquisition d'un ordinateur d'occasion. Rien ne permet de garantir que le disque n'a pas été maltraité.

Alors si vous voulez installer un système ou des données tout de suite, ne vous gênez pas mais vous détecterez peut-être le problème quand il sera trop tard. Détendez-vous et lancez quelques tests.

Pour commencer, l'utilitaire du constructeur ainsi que `SMARTUDM.EXE` (présents sur The Ultimate Boot CD) élagueront les problèmes sans interférence avec un *kernel*. Puis vous pourrez lancer `badblocks` avant de partir en week-end.

Acte II : on va bien s'ennuyer !

Un peu de patience

`badblocks` est l'étape indispensable avant l'utilisation d'une mémoire de masse. Toute mémoire de masse, même les mémoires Flash ! (J'ai eu le cas d'une clé USB dont la moitié de la capacité était défectueuse). Tout(e) informaticien(ne) devrait être familiarisé(e) à son utilisation, qui est décrite dans sa page *man*. Pour les débutants, une introduction est fournie dans cet article.

C'est un utilitaire bas niveau et donc potentiellement dangereux. Il faut donc bien comprendre son fonctionnement et ne pas l'utiliser à moitié endormi.

Dans le cas qui nous intéresse ici, le mode de test utilisé est ***écriture*** (option `-w`). D'une part, le disque étant neuf, il n'y a aucun risque de perdre des données par surécriture. Essayez juste de taper le bon numéro de partition pour ne pas détruire un autre disque...

Une autre raison est que le mécanisme de relocation dynamique des secteurs défectueux doit être relancé. Une lecture seule détectera les fautes, mais l'écriture va les corriger, car le contrôleur saura quelles données (valides) mettre dans le secteur déplacé.

Si l'écriture n'est pas effectuée, le disque dur renverra encore une erreur la prochaine fois que le secteur endommagé sera lu. Lorsqu'on lui envoie une donnée à écrire, cette donnée ira sur un autre secteur « de réserve » et la prochaine lecture sera juste déroutée par le contrôleur vers le nouvel emplacement de la donnée.

Ce mécanisme est mis aussi en valeur dans la documentation de `smartctl`[1].

Au cas où vous ne connaissez pas cette commande, `badblocks -w` effectue quatre passes consistant en une écriture puis une lecture chacune. C'est suffisant pour que le contrôleur intégré trouve et corrige tous les défauts.

Le PC, lui, ne fait rien d'autre que pomper des données dans le vide, on peut jouer à Frozen Bubble en attendant. Les données écrites sont 4 constantes différentes, qui sont relues et vérifiées les unes après les autres.

Pour éviter que la mémoire cache d'écriture du disque dur n'interfère, il faut choisir une taille de bloc de test très grande. Pour un disque récent avec 8MO de cache, il faut utiliser des passes d'au moins 16MO.

En pratique, la mémoire cache d'écriture n'est pas activée afin de limiter les dégâts en cas d'arrêt brutal (coupure de courant accidentelle ou non, plantage du système...) : des données critiques pour le système de fichiers pourraient disparaître et rendre le reste du disque inutilisable.

Mais le cache de lecture est normalement actif, donc il faut encore se méfier.

Seulement, voici le vrai hic : `badblocks` a été conçu pour les disques durs et disquettes de l'époque.

Les modes de fonctionnement de badblocks

Ce programme est relativement rustique et n'intègre pas d'intelligence. Sa fiabilité dépend donc de son utilisation.

Appelé sans option, `badblocks` se contente de lire toute la zone indiquée (disque ou partition par exemple). C'est un test rapide (une seule passe) et sans risque mais qui peut laisser passer certains types d'erreur comme des bits « collés » (toujours à 1 ou à 0). On peut cependant l'utiliser sur une partition déjà utilisée mais soupçonnée d'être endommagée.

Pour être sûr que le média ne présente pas de défaut, il existe un mode destructif avec écriture et lecture. L'option `-w` effectue quatre passes successives pour vérifier plusieurs organisations de bits :

```
10101010 (0xAA)
01010101 (0x55)
11111111 (0xFF)
00000000 (0x00)
```

Chaque *pattern* est écrit sur l'ensemble de la zone indiquée, puis relu pour vérifier qu'aucune erreur n'est apparue. Ce test est destructif et ne doit s'effectuer, au pire, que lorsque la perte de données est possible. Il faut donc l'utiliser avec prudence. Ce test a été conçu pour les disquettes et les disques durs antiques où chaque bit avait une représentation séparée.

L'altération du support pouvait donc signifier qu'un bit est toujours à 1 ou à 0, ce qui explique les patterns `00000000` et `11111111`. Les patterns `01010101` et `10101010` servent pour vérifier l'immunité de l'encodage de type « MFM » ou « Manchester », où c'est la position des transitions des signaux qui détermine la valeur des bits.

Cela n'a plus d'utilité actuellement car les encodages récents sont extrêmement sophistiqués et des codes de correction d'erreurs multiples permettent de protéger des grands groupes de bits ou même des secteurs entiers.

Des méthodes comme « Reed-Solomon entrelacé » (utilisé aussi pour les CD) et des signatures de contrôle générées par le matériel (sans intervention du noyau) permettent de détecter et corriger les erreurs dues à des impuretés microscopiques qui affectent un ou plusieurs bits.

Les patterns ci-dessus ne sont donc plus efficaces aujourd'hui : le disque dur indique simplement si la lecture d'un secteur a réussi, au bout d'un temps proportionnel à la difficulté de reconstruire le secteur. En cas d'erreur de lecture, le disque va d'abord retenter plusieurs fois de lire le secteur pour glaner des informations complémentaires.

Un signal d'échec renvoyé par le disque signifie qu'il y a trop d'impuretés ou que la surface a été endommagée (frottement de la tête sur le revêtement magnétique causé par des vibrations par exemple) et que les données ne sont pas reconstructibles. C'est ce dernier point qui nous intéresse dans cet article.

Enfin, l'option `-n` effectue des tests d'écriture et de lecture mais en sauvegardant les données initiales pour les restaurer ensuite, ce qui est donc non destructif. Cela permet de confirmer des doutes émis lors d'un test en lecture seule. Dans les deux derniers cas, le programme a besoin d'un espace temporaire en mémoire pour contenir les blocs traités. En sélectionnant une taille de bloc trop grande, la mémoire de l'ordinateur risque de s'épuiser.

Attention : `badblocks` est normalement appelé directement par `mke2fs` pour vérifier la surface de la partition à formater. Si les scripts présentés ici persistent à indiquer des secteurs défectueux, il faudra adapter la taille des blocs et fournir la liste (corrigée) des secteurs à `mke2fs`.

Il faut aussi noter que `badblocks` est sensible à toutes les erreurs de données, même si elles ne viennent pas du disque dur. Par exemple, un câblage de mauvaise qualité, des interférences ou (pourquoi pas) des particules ionisantes (issues du cosmos ou d'une désintégration nucléaire d'un atome proche) peuvent changer la valeur d'un bit sur le bus de données (à votre avis, pourquoi les ordinateurs ont-ils aujourd'hui des mémoires et des bus à correction d'erreur ?). Ce changement sera aussi détecté par `badblocks`.

En règle générale, il faut donc se préoccuper autant des éventuels faux positifs que des faux négatifs. Pour cela, même si c'est une méthode très lourde, il est préférable de vérifier un disque dur avec l'option `-w` qui effectue plusieurs passes, « rodant » ainsi l'ensemble du disque et du chemin de donnée dans l'ordinateur.

Il va vaillamment balayer sans broncher (pas comme vous) tout le disque. S'il a une capacité de 250GO, ce sera ... long.

Petit calcul du temps nécessaire : `(250GO x 8) / 30MO/s= environ 70000s ou 18 heures`. Le débit de 30MO/s est hypothétique mais réaliste.

Il faut donc être patient ! Surtout que d'autres soucis doivent être résolus.

En particulier, lors des quatre passes, un secteur détecté comme défectueux n'est pas rebalayé par une passe suivante, son adresse est juste mise dans une liste qui est affichée à la fin du programme ou envoyée dans un fichier.

Or cela ne permet pas de vérifier que la relocation du secteur concerné a correctement eu lieu (ou que le problème est juste transitoire).

L'autre problème est que des erreurs vont donc probablement être affichées à la fin du balayage complet.

Cela varie de disque à disque mais c'est hautement probable. La conséquence est qu'il faut rebalayer le disque si des secteurs défectueux sont détectés.

La patience a des limites, le disque devra été balayé 16 fois en l'espace de 36 heures. Des mesures sont à prendre !

Et tant qu'à faire, il serait désirable de pouvoir interrompre le processus et de le continuer plus tard, au cas où l'ordinateur devait servir à autre chose temporairement. Ou s'il plante.

Un peu de tuning

Pour éviter de perdre bêtement du temps, il est important de vérifier que le système travaille à sa fréquence maximale.

L'outil `hdparm` permet de mesurer la vitesse maximale de transfert du disque et de régler certains paramètres du noyau ou du contrôleur de disques durs.

En premier lieu, `hdparm /dev/hdX` retourne la configuration courante et les paramètres du disque `X`.

Vérifiez que la DMA est activée, que le transfert s'effectue en mode 32 bits, etc.

Par exemple :

```
root:~# hdparm /dev/hda

/dev/hda:
 multcount    = 16 (on)
 I/O support  =  0 (default 16-bit)
 unmaskirq    =  0 (off)
 using_dma    =  1 (on)
 keepsettings =  0 (off)
 nowerr       =  0 (off)
 readonly     =  0 (off)
 readahead    =  8 (on)
 geometry     = 3648/255/63, sectors = 58605120, start = 0
 busstate     =  1 (on)
```

Pour vérifier que les paramètres ont pris effet, lancez `hdparm -T -t /dev/hdX` avant et après les changements de chaque paramètre. Sur mon ordinateur portable, j'obtiens :

```
root:~# hdparm -T -t /dev/hda

/dev/hda:
 Timing buffer-cache reads:  128 MB in  1.13 seconds =113.27 MB/sec
 Timing buffered disk reads:  64 MB in  3.33 seconds = 19.22 MB/sec
```

C'est la dernière ligne, donnant le débit en pointe du disque, qui compte. Activer la DMA (option `-d 1`) et le mode ***UltraDMA*** (`-X 66/67/68`...) donne en général de bons résultats mais le kernel devrait le faire automatiquement au démarrage. Les options `idebus=66 hda=autotune hdb=autotune ...` donnée au noyau lors du démarrage devraient l'aider selon les cas.

Suivant votre distribution GNU/Linux, `hdparm` est probablement exécuté au démarrage du système. Le fichier de configuration varie d'une distribution à l'autre, mais jetez un oeil attentif dans `/etc/`.

En effet, les modifications de comportement du disque dur opérées avec `hdparm` seront perdues au prochain redémarrage, il faudra modifier ou configurer des fichiers de votre système d'init pour qu'elles soient permanentes.

Un peu de mesure

Le débit réel, lui, doit être mesuré de manière plus pragmatique. Les débits retournés par `hdparm -T -t /dev/hdX` sont des pointes ne tenant pas compte de l'*overhead* du système d'exploitation et ne sont mesurées que pour la lecture.

La différence entre la vitesse d'écriture et de lecture peut atteindre 30% et les débits varient selon la position de la tête de lecture, car on peut mettre plus de données au début qu'à la fin du disque. Mais cela varie entre les modèles, le comportement peut être surprenant.

Il ne faut pas non plus oublier que les interfaces modernes super-rapides (ATA133, devant permettre des transferts à 133Mo/s, ce qui représente tout de même un milliard de bits par seconde) ne sont pas exploitées au maximum de leurs capacités. La tête de lecture ou une autre partie du disque dur va rapidement atteindre ses performances maximales, et les autres circuits sont dimensionnés en fonction de ce maximum.

Enfin, lors de la lecture d'un secteur réalloué, la tête de lecture fera un écart, réduisant ainsi la bande passante.

Pour mesurer le débit réel, il suffit d'ouvrir un deuxième terminal et de lancer la commande `vmstat 1`. Les colonnes `bo` (kBytes Out) et `bi` (kBytes In) indiquent le nombre de kilooctets transférés entre deux lignes. L'option `1` indique qu'une nouvelle ligne est affichée après une seconde. Nous mesurons donc le débit en kilooctets par seconde. CQFD.

la lecture anticipée et de la mémoire cache (remplie) du disque. Lorsque `badblocks` sera en action, cette méthode permettra de surveiller son fonctionnement, aussi bien en lecture qu'en écriture.

Fort heureusement, `badblocks` effectue des transferts de très longs blocs contigus. Il y a donc peu de pertes de temps pour positionner la tête de lecture et les mesures sont assez cohérentes.

Les débits mesurés seraient bien différents si on avait lancé une compilation du noyau, à cause de nombreux accès désordonnés et de l'effet des différents niveaux de mémoire cache.

Un peu d'optimisation

Le système, même au mieux de sa forme, va encore devoir mouliner un bon bout de temps. Les vitesses de transfert augmentent moins vite que les capacités des disques durs. Pourtant, il faudra bien le balayer intégralement, ce disque !

Fenêtre 1

```
root:~# vmstat 1
  procs                  memory                      swap                io                system                cpu
r   b   w   swpd     free     buff    cache           si        so        bi        bo        in    cs   us   sy   id
0   0   1      0   141736     1040     5036            0         0        39         0       107    18    0    0  100
0   0   0      0   141736     1040     5036            0         0         0         0       101     7    0    0  100
...
```

Fenêtre 2

```
root:~# cat /dev/hda > /dev/null
([Ctrl]+[C] après quelques secondes)
```

Fenêtre 1

```
...
0   0   0      0   141712     1064     5036            0         0         0         0       105    16    0    0  100
0   0   0      0   141712     1064     5036            0         0         0         0       105    14    0    0  100
0   0   0      0   141704     1072     5036            0         0         0        88       119    17    0    0  100
1   0   0      0   132980     1072    13680            0         0      8644         0       239   280    0    7   93
1   0   0      0   113140     1072    33520            0         0     19840         0       414   627    0   17   83
1   0   0      0    92788     1072    53872            0         0     20352         0       423   642    0   13   87
1   0   0      0    71764     1072    74480            0         0     20608         0       422   645    0   14   86
1   0   0      0    49512     1080    94704            0         0     20224        20       427   656    2   12   86
0   1   0      0    26560     1080   115568            0         0     20864         0       444   686    0   17   83
0   0   0      0   135484     1080     5036            0         0     16896         0       373   543    0   20   80
0   0   0      0   135484     1080     5036            0         0         0         0       102     8    0    0  100
0   0   0      0    35484     1080     5036            0         0         0         0       105    13    0    3   97
([Ctrl]+[C] pour arrêter)
```

On observe un débit soutenu de 20MO/s (pas mal pour un vieux *laptop*), confirmant la mesure de `hdparm` en lecture. La différence avec la mesure effectuée par `hdparm` est qu'il mesure le temps total du transfert, alors que nous venons de prendre un instantané au milieu du processus. `Hdparm` mesure donc aussi un *overhead* lié au démarrage du transfert, alors que la mesure par `vmstat`, bien que légèrement moins précise, tient compte de

On peut cependant éviter les opérations inutiles ou redondantes, en particulier lors du deuxième balayage, où on veut seulement vérifier que les éventuels secteurs défectueux ont été correctement « corrigés ».

La solution que j'ai trouvée à ce problème consiste à créer un script en *bash* pour automatiser des vérifications partielles et assurer le redémarrage après une interruption.

Cette fonctionnalité étant d'autant plus importante qu'un ordinateur de mon parc a tendance à planter aléatoirement durant de longues vérifications... (il semblerait qu'il s'agit d'un problème de kernel)

Pour chaque partie du disque, `badblocks` va créer un fichier contenant la liste des secteurs défectueux qu'il aura trouvé. Avec des blocs suffisamment petits, la majorité des fichiers sera vide. Le script va d'abord vérifier que le fichier correspondant à la partie en cours est absent, avant de lancer `badblocks`. Pendant le premier lancement, tous les fichiers seront créés. Il suffit ensuite d'examiner et éventuellement de supprimer les fichiers non vides avant de redémarrer le script, pour qu'il ne s'occupe que des parties posant des problèmes.

En raison de la tendance du système à planter, j'ai placé le script et les fichiers de sortie dans une clé USB et le système tournait avec un « LiveCD ». J'ai donc ordonné une resynchronisation des disques (`sync`) après chaque fin de `badblocks` car le système de fichiers FAT n'est pas réputé pour sa tolérance aux arrêts brutaux.

```
00 # Fichier /mnt/win_c3/bad.sh
01 # script  pour scanner un disque de 160G
02 # au cas où l'ordinateur plante
03 # pathologiquement aléatoirement.
04 # l'OS est une MandrakeMove 9.2,
05 # le disque dur est sur /dev/hde et
06 # les "logs" sont dans /mnt/flashdisk/blks/
07 #
08 # quelques calculs :
09 # taille d'un bloc hdparm : 4096 (4K)
10 # taille d'une passe : 4K*10000 blocs = 40M
11 # taille d'une étape badblocks : 40M*100=4G
12 # taille du disque : 4G * 40 passes = 160G
13
14 dev=/dev/hde
15
16 n=/mnt/flashdisk/blks/block0
17 if [ ! -f $n ] ; then
18   echo --------------------------------
19   echo BLOCK $i
20   badblocks -b 4096 -c 10000 -s -v -w \
21     -o $n $dev  999999  000000
22   sync
23 fi
24
25 for i in $(seq 1 39)
26   do
27     n=/mnt/flashdisk/blks/block$i
28     if [ ! -f $n ] ; then
29       echo --------------------------------
30       echo BLOCK $i
31       badblocks -b 4096 -c 10000 \
32         -s -v -w -o $n $dev \
33         $(echo -n $i ; echo -n 999999) \
34         $(echo -n $i ; echo -n 000000)
35       sync
36     fi
37   done
38
39 n=/mnt/flashdisk/blks/block40
40 if [ ! -f $n ] ; then
41   echo --------------------------------
42   echo BLOCK 40
43   badblocks -b 4096 -c 10000 -s -v -w \
44     -o $n $dev 40209120 40000000
45   sync
46 fi
47
48 echo "fini !"
```

Comme on peut le voir, la taille du disque et d'autres facteurs rendent ce script plus complexe que nécessaire. Evidemment, un UNIX digne de ce nom permet de s'en sortir avec peu d'efforts et avec de nombreuses méthodes. J'ai choisi la plus simple, sans même passer par Python ou Perl, grâce à une petite astuce et quelques calculs. La mise au point m'a pris peut-être dix minutes, en comptant des révisions de la syntaxe de Bash. Ce n'est pas un modèle de portabilité ni même de propreté mais il fonctionne, ce qui est un argument suffisant pour l'utiliser ;-)

Le coeur du programme est évidemment la boucle.

```
25 for i in $(seq 1 39)
26   do
  ....
37   done
```

Qui s'exécute avec `i` variant de 1 à 39. Il existe d'autres moyens plus propres de boucler avec bash, comme cette syntaxe plus proche du C :

```
for (i=1; i<40; i++)
do
 ...
done
```

La première mauvaise astuce consiste à se passer de calculs en effectuant la multiplication « à la main » en concaténant des zéros à la suite du compteur.

La concaténation est effectuée par `$(echo -n $i ; echo -n 000000)` et `$(echo -n $i ; echo -n 999999)` car écrire `$i000000` ou `$i999999` n'aurait pas donné le résultat attendu, contrairement à l'expression `n=/mnt/flashdisk/blks/block $i` qui crée le nom du fichier de sortie. Ce n'est peut-être pas très lisible mais je n'avais pas pris le temps de lire tous les détails concernant l'évaluation des expressions mathématiques par bash.

Un peu de calcul mental

Avant de lancer le script, il faut calculer la taille des différents types de blocs en fonction de plusieurs paramètres.

- `badblocks` travaille par défaut sur des blocs de 4096 octets (j'ai utilisé l'option `-b 4096` pour m'en assurer).
- Le disque dur contient 160 milliards d'octets.
- La mémoire cache du disque dur, donc la moitié de la taille minimale d'une passe, est de 2MO .
- Il serait sage de limiter le nombre de fichiers de sortie à moins de 100 pour ne pas trop encombrer la petite clé USB.

A partir de ces données, j'en ai déduit les paramètres indiqués en début du script. Il faut aussi tenir compte du fait que `badblocks` travaille aussi en plusieurs étapes (pour la barre de progression), dont il faut indiquer la taille (ici avec le paramètre `-c 10000` donc une passe couvre 10000 blocs).

Avec des passes de 4096×10000 octets, soit 40MO, la mémoire cache du disque dur ne peut pas causer de faux résultats positifs (si elle est activée).

A un niveau supérieur, il faut indiquer le numéro du bloc de fin et d'arrivée (allez comprendre pourquoi c'est inversé). L'unité est sur 4KO, il y a donc `160G/4k=40M` blocs, ou plus précisément `40209120`. Le nombre exact est donné par `badblocks` lui-même, en le lançant succinctement, puisqu'il affiche les numéros de blocs de début et de fin.

On veut exécuter `badblocks` moins d'une centaine de fois, avec un nombre de blocs qui est une puissance de 10 (pour utiliser l'astuce de la concaténation). Avec 40 itérations, chaque partie analysée fait `160/40=4GO` ou 100 passes de 40MO. Vous suivez ?

Un peu de bricolage

Il y a deux effets de bord. Pour commencer, `badblocks` n'aime pas le résultat de la concaténation lors de la première itération, lorsque la chaîne retournée (`0999999`) commence par un zéro. J'ai donc copié-collé-modifié le corps de la boucle pour écrire certains nombres en dur.

Le deuxième problème survient à la fin. Je croyais que `badblocks` s'arrêterait tout seul de balayer le disque à la fin de celui-ci mais il a continué en créant une liste gigantesque de secteurs défectueux.

Comme pour le premier problème, dont la solution fonctionne, j'ai recopié puis modifié le corps de la boucle vers la fin pour insérer la taille réelle du disque. Voici donc révélées toutes les raisons de l'aspect peu banal de ce script ! Libre à vous de l'adapter ou de le traduire dans un autre langage.

A l'usage, il s'est révélé très utile, deux disques de 160GO ont ainsi été vérifiés en douze heures chacun. Les quelques

secteurs défectueux ont disparu à la deuxième passe. Mission accomplie !

Je n'ai pas automatisé l'effacement des fichiers signalant des erreurs car je préfère les examiner moi-même. Une boucle `while` à l'intérieur de la boucle for aurait fait l'affaire mais le comportement n'aurait probablement pas été bon si de grosses erreurs étaient découvertes (entraînant, par exemple, la saturation de la clé USB).

Et puis, il faudrait créer une fonction pour éviter la duplication du code de la boucle. Je vous laisse le nettoyage comme exercice, car ce n'est qu'une mise en bouche :-)

Acte III : accrochez-vous !

Un peu d'épouvante

Je viens de décrire un cas normal, commun et qui se termine bien. Le script compense l'absence de l'option formatage bas niveau dans l'utilitaire du constructeur du disque, et permet le redémarrage après une interruption.

Mais parfois, tout part en vrille et il faut intervenir manuellement et très fermement.

En remplacement d'un disque dur 2"5 endommagé, le service après-vente du constructeur (en passant par le magasin vendeur, ce qui a pris... beaucoup de temps) m'a fourni un disque dur de même capacité mais avec une jolie étiquette annonçant fièrement la couleur : « **SERVICEABLE USED PART** ».

Procédure classique : lancement de `hdparm` puis `badblocks` (d'une traite, le disque étant relativement petit). Le résultat est sans appel : le disque « gratte » au bout de quelques secondes et le noyau affiche des injures.

La bonne nouvelle est que la zone de la table des partitions est intacte, ce qui permet de partitionner le disque. La mauvaise nouvelle est qu'il est difficile de connaître l'étendue de la partie endommagée.

Ironiquement, il n'est même pas possible de lancer `hdparm -t` car le test s'effectue sur les premiers 64MO du disque, alors que la première zone valide semble ne s'étendre que sur 30MO environ. Il est donc impossible de tuner le disque.

A force de tâtonnements successifs, en partitionnant puis en vérifiant les partitions créées, j'ai circonscrit une zone intacte au début et à la fin du disque. Mais il a une capacité de 20GO !

Un peu d'aide extérieure

Par l'intermédiaire de forums et de recherches sur Internet, j'ai été aiguillé vers les utilitaires de diagnostic développés par les constructeurs.

Une liste et de précieux conseils sont réunis sur une page du site **http://www.hardware.fr** datant du 25 janvier dans la rubrique *Hardware/HDD* : **http://forum.hardware.fr/hardwarefr/Hardware/sujet-605153-1.htm**

IBM Drive Fitness Tool, desormais HGST (Hitachi Global Storage Technologies) : **http://www.hgst.com/downloads/DFT32-V340.EXE**

Maxtor MaxDiag : **http://www.maxtor.com/en/support/downloads/files/powermax.exe**

Seagate SeaTools : **http://www.seagate.com/support/npf/seatools/seatoold.exe**

Western Digital Datalifeguard : **http://support.wdc.com/download/dlg/dlginstall_10_0.exe**

Des noctamoules m'ont aussi indiqué un projet : The Ultimate Boot CD à **http://ultimatebootcd.com** qui regroupe sur un même CDROM les utilitaires ci-dessus ainsi que des dizaines d'autres tout aussi utiles.

Certains utilitaires sont basés sur un système FreeDOS (libre) ou PC-DOS (IBM), d'autres sur Linux, mais tous cohabitent car le CDROM est une collection d'images de disquettes que l'on peut lancer par l'intermédiaire d'un simple menu.

Après avoir chargé les fichiers sur les sites, démarré l'ordinateur avec, lancé l'utilitaire du constructeur, le verdict est sans appel : il est considéré comme irrémédiablement endommagé, à remplacer, *kaputt*.

```
Failure Code : 0x72     Defective Device. S.M.A.R.T. Failure.
```

Il y a probablement tellement de secteurs défectueux qu'il n'y a plus assez de place dans la zone de secours.

Fait étrange, en pleine manipulation, la géométrie du disque reportée par `fdisk` a même changé. La granularité a fait un bond, le partitionnement devient alors moins précis. Les cylindres grossiers entraînent des pertes de place si un secteur seulement est inutilisable. La première zone valide de 37MO a donc rétréci. Pourtant le disque n'est pas à jeter : il y a au moins plusieurs Gigaoctets qui ont été vérifiés comme corrects vers la fin du disque. Et il doit y avoir un moyen de forcer le nombre de têtes.

Mais il est impossible de lancer `badblocks` normalement car chaque secteur trouvé comme défectueux bloque l'ordinateur pendant plusieurs minutes, durant lesquelles le noyau tente de relancer la lecture. Si on insiste trop, le disque refuse de répondre et se bloque, forçant un redémarrage de l'ordinateur.

A raison de deux minutes par secteur, le balayage total du disque serait réellement interminable et il y a des mégaoctets de secteurs morts. Sans compter les bruits d'agonie qu'il émet.

Un peu plus de bash

Cette fois-ci, il faut employer des mesures plus radicales tout en tenant compte des limitations de la plateforme. Le seul « liveCD » qui peut fonctionner sur le *laptop* contenant le disque dur est en fait le CDROM d'installation de la distribution Slackware 10.0.

Cette distribution n'utilise pas d'interface graphique ou de script pour le formatage des disques, mais `busybox` avec un kernel minimal qui ne plante pas lors de la détection du port PCMCIA (problème résolu depuis par une option dans le BIOS), tout en reconnaissant l'USB. Un nombre restreint d'outils en ligne de commande est donc disponible autour d'un *shell* et avec le support du clavier français, s'il vous plaît. Presque l'idéal.

La vérification du disque s'effectue toujours avec d'une part le CDROM « pseudo-live », et d'autre part une clé USB contenant le script suivant et quelques binaires exécutables absents du système minimal. Par exemple, `gpm` (gestion de la souris en mode texte, très pratique pour faire du copier-coller entre deux terminaux) et `nano` (un éditeur simple mais tellement plus intuitif que vi) ont été indispensables pour développer les scripts.

Mais surtout, bash est présent car il peut analyser des expressions plus complexes que le shell par défaut. C'est particulièrement utile car il va falloir faire des vrais calculs sur les numéros de secteurs. On va pouvoir oublier les bricolages de concaténation, qui entraînent les effets de bord du premier script.

L'autre astuce consiste à lancer badblocks sur seulement quelques secteurs par cylindre. Badblocks permet d'indiquer les numéros de secteurs de début et de fin pour le balayage, et n'autorise pas des nombres identiques, donc on indique deux nombres consécutifs. Chaque lancement préliminaire de badblocks va donc tester deux secteurs.

Ainsi, si le cylindre entier est défectueux, on ne perd pas trente-trois mille heures à le balayer. Si plusieurs secteurs à différents endroits du cylindre sont bons, on peut lancer badblocks normalement sur le cylindre entier.

A la fin, on veut obtenir les numéros de cylindre de début et de fin de toutes les zones intactes, afin de ne pas faire de calcul hasardeux pour partitionner avec fdisk. D'ailleurs, on obtient les informations sur la géométrie du disque (C:H:S) avec ce même programme :

```
root@slackware:/d# fdisk -l /dev/hda

Disk /dev/hda: 20.0 GB, 20003880960 bytes
255 heads, 63 sectors/track, 2432 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Disk /dev/hda doesn't contain a valid partition table
```

Un peu de cambouis sur les bras et le visage

Une autre méthode existe, si fdisk n'est pas disponible :

```
root@slackware:/# cd /proc/ide/hda
root@slackware:/proc/ide/hda# cat geometry
physical    38760/16/63
logical     10/10/10
```

Comment obtenir cette géométrie logique bizarre ? J'ai simplement donné ce paramètre de démarrage au noyau :

```
boot: bare.i hda=10,10,10
(au format C:H:S)
```

Le répertoire /proc/ide/hda contient de nombreuses informations sur /dev/hda, par exemple :

```
root@slackware:/proc/ide/hda# cat capacity
39070080
(nombre de secteurs de 512 octets)
```

Cette méthode d'accès est aussi une alternative à hdparm par l'intermédiaire du fichier settings. Par exemple, pour activer la DMA :

```
root@slackware:/proc/ide/hda# echo using_dma:1 > settings
```

La liste des options est disponible simplement en lisant le fichier settings. Leur nom est assez parlant mais il faut quand même faire attention.

Ces techniques permettent de corriger le petit incident du changement impromptu de la géométrie. Ainsi, on peut réduire la taille des cylindres et cerner les zones endommagées au plus près, ce qui réduit l'espace perdu lors du partitionnement.

```
root@slackware:/d# fdisk -l /dev/hda

Disk /dev/hda: 20.0 GB, 20003880960 bytes
10 heads, 10 sectors/track, 390700 cylinders
Units = cylinders of 100 * 512 = 51200 bytes

Disk /dev/hda doesn't contain a valid partition table
```

On voit que fdisk n'est pas un programme si rudimentaire car il a corrigé lui-même le nombre de cylindres erronés.

L'autre méthode de modification de la géométrie, « en live », accède directement au fichier settings :

```
root@slackware:/proc/ide/hda# echo bios_cyl:38760 > settings
root@slackware:/proc/ide/hda# echo bios_head:63   > settings
root@slackware:/proc/ide/hda# echo bios_sect:16   > settings
```

Ce qui configure la géométrie logique de manière identique à la géométrie physique indiquée plus haut par le fichier geometry

En fait, la géométrie physique ou logique a relativement peu d'importance car la plupart des disques sont accédés en mode LBA, c'est à dire avec l'adresse linéaire des secteurs, sans numéro de secteur par piste, de têtes ou de cylindres. Dans l'ombre, le contrôleur intégrés dans le disque dur fait lui-même la traduction d'adresse. Les questions de géométrie (paramètres C.H.S.) apparaissent seulement lors du partitionnement et une configuration exotique peut être facilement indiquée au noyau par un paramètre de démarrage, comme nous venons de le voir.

Les autres options sont décrites dans votre répertoire de documentation : **/usr/src/linux/Documentation/ide.txt**

Arrachage de cheveux

Le plan semble simple : on veut que le script teste quelques paires de secteurs, puis tout le cylindre si le test est concluant. Il devra afficher des paires de numéros de cylindre, correspondant au début et à la fin des zones valides.

Mais les premiers tests montrent que rien ne se passe comme prévu. Lors du premier test d'une paire de secteurs connus pour être défectueux, le disque se met à gratter et à crépiter, le noyau s'inquiète et relance le disque dur, pour arriver quinze secondes plus tard à un résultat correct. Le disque dur a réalloué les deux secteurs !

Ce n'est pas ce qui était prévu et le « faux positif » oblige à changer d'approche. Si le résultat retourné par badblocks est mauvais, il faut trouver un autre moyen.

On peut explorer deux possibilités : la mesure du temps d'exécution et la capture des messages du noyau, toutes deux indiquant que le disque dur rencontre des difficultés.

En ce qui concerne le chronométrage, il est peu pratique de réutiliser la commande time intégrée à bash car le format de sortie est trop complexe à *parser*. Mais au détour d'une relecture de la page man, j'ai découvert (on en apprend tous les jours !) la variable d'environnement $SECONDS qui est incrémentée (comme son nom l'indique) toutes les secondes.

L'utilisation est d'une facilité déroutante, à mitiger avec les incertitudes et l'imprécision inhérentes à cette méthode.

```
SECONDS=0
 # commande à mesurer
echo $SECONDS
```

La variable $SECONDS étant incrémentée automatiquement toutes les secondes, il faut la recopier avant de l'utiliser plusieurs fois (pour éviter les *race conditions*). Mais elle ne sert normalement que comme opérande pour une opération conditionnelle.

```
if (( $SECONDS > 3 ))
then
  echo "échec au bout de $SECONDS secondes"
fi
```

Dans la pratique, je laisse trois secondes au programme, au cas où il serait lancé lorsque le disque est au repos ou si les secteurs sont relogés mais valides. De plus, je ne suis pas sûr (c'est indiqué nulle part) si l'assignation remet aussi le *timer* d'incrémentation à zéro. L'incrémentation peut donc se produire moins d'une seconde après l'assignation, il faut

donc s'attendre à une imprécision d'une seconde, ce qui justifie d'autant plus la marge confortable.

L'autre méthode consiste à appeler `dmesg` pour lire les messages du noyau. Sans paramètre, la commande affichera juste le tampon des messages, qui s'y accumulent tant qu'il y a de la place. Ce n'est pas très pratique puisqu'il faut juste savoir si des erreurs se sont produites durant une période spécifique.

Là encore, une option se cache dans la page man : avec `-c`, le tampon est vidé après l'affichage. Il faut donc lancer

```
dmesg -c > /dev/null
avant chaque essai, puis

msg=$(dmesg -c)
if [ -n "$msg" ]
then
   ...
fi
```

pour vérifier que tout s'est bien passé.

Au final, les vérifications d'un secteur s'effectuent à l'intérieur de la fonction `verifier ()` pour factoriser un peu les opérations et réduire la taille du script.

```
01 function verifier () {
02 # argument : commande à effectuer
03 echo ' * commande : ' $@
04 SECONDS=0              # remet le compteur à zéro
05 dmesg -c >> /dev/null # vide la liste des erreurs
06 if $@                 # exécute la commande
07   then
08     echo temps = $SECONDS s.
09     if (( $SECONDS > 3 ))
10     then
11       echo échec : trop long !
12       return 1
13     fi
14     if [ $(dmesg) ] # vérifie qu'aucune erreur n'est survenue
15     then
16       echo ' --- échec : dmesg renvoie : ---'
17       dmesg
18       return 1
19     fi
20   else
21     echo ' --- la commande a échoué. ---'
22     return 1
23   fi
24
25   return 0
25 }
```

On peut noter que les deux tests (chronométrage et erreurs du noyau) fonctionnent tous bien en pratique. On pourrait n'en utiliser qu'un seul, mais on n'est jamais trop prudent.

Calvitie

Comme si cela n'était pas déjà suffisamment compliqué, le kernel s'obstine à vouloir lire 16 secteurs consécutifs. Si on tombe sur une zone « morte », il faut deux minutes pour que le disque dur se calme et arrête de crépiter.

La première idée est de remplacer `badblocks` par `dd`

```
# test en lecture
verifier "dd if=$dev of=/dev/null count=1 bs=512 skip=$1" &&
# test en écriture
verifier "dd if=/dev/null of=$dev count=1 bs=512 seek=$1"
```

mais le comportement est le même. `dd` permet de limiter la recherche à un seul secteur au lieu de deux, mais le kernel continue à accéder à 16 secteurs.

Pour modifier ce comportement, on se tourne vers hdparm :

```
hdparm -a 0 -A 0 -m 0 $dev
```

- a 0 : nombre de secteurs pour la lecture anticipée.
- A 0 : désactive la lecture anticipée.
- m 0 : nombre de secteurs par requête.

La désactivation de la lecture spéculative des secteurs n'a aucune incidence non plus. En fouillant dans /proc/ide/hda, on peut aussi tenter ceci

```
root@slackware:/proc/ide/hda# echo max_failures:0 > settings
```

(la valeur de max_failures est initialisée à 1 au démarrage). Mais ce changement n'a pas plus d'effet. Le problème se situe encore plus bas dans le noyau.

Pour les plus aventureux, il est possible de modifier le comportement du kernel en modifiant le fichier /usr/src/linux/include/linux/ide.h, en particulier les lignes :

```
#define ERROR_MAX    8  /* Max read/write errors per sector */
#define ERROR_RESET  3  /* Reset controller every 4th retry */
#define ERROR_RECAL  1  /* Recalibrate every 2nd retry */
```

Mais recompiler un kernel spécialement pour un disque dur est une mesure impossible à mettre en oeuvre dans la plupart des cas et cela ne répond pas à la demande qui est aussi de ne tester qu'un seul secteur à la fois.

Par dépit, on peut encore utiliser la fonction suivante et attendre un peu :

```
01 dev=/dev/hda
02
03 function test_secteur () {
04   if [ "$1" ] # il faut un argument : le n° du secteur
05   then
06     # désactive la lecture groupée des secteurs
07     hdparm -d 0 -m 0 -a 0 -A 0 $dev
08
09     verifier  "badblocks -b 512 -c 1 -s -v    $dev $(( $1+1 )) $1" &&
10      verifier "badblocks -b 512 -c 1 -s -v -w $dev $(( $1+1 )) $1" &&
11       return 0 # succès
12
13     return 1
14   else
15     echo "il manque un argument"
16     return 1
17   fi
18 }
```

Un peu d'espoir

Je ne suis pas le seul à m'être confronté au problème des tampons de Linux et Google aiguille vers plusieurs discussions relatives sur des forums.

La solution semble s'appeler O_DIRECT. C'est un paramètre d'appel de la fonction open(2), introduit récemment dans le noyau Linux. La documentation de la glibc n'est pas encore à jour et c'est encore Google qui fournit les informations vitales.

En résumé, O_DIRECT indique que les données du fichier ainsi ouvert ne transiteront pas par un tampon dans le noyau lors des lectures et écritures. Il n'y a pas de recopie des données dans l'espace du noyau, le transfert s'effectue directement entre l'espace utilisateur et le disque.

La motivation de l'introduction de ce mode est principalement la réduction de l'overhead du noyau et donc de la latence pour des applications « temps-réel » ou les bases de données. Comme le noyau ne gère alors plus de tampons de fichiers, qui sont donc normalement de 8 ou 16 secteurs, il serait possible de faire un petit programme qui va juste lire un secteur sans que le noyau n'envoie une requête plus grosse au disque.

En contrepartie de l'absence de tampon, permettant de gérer des accès de toutes tailles et non alignés, le noyau expose les limitations de l'interface accédée et les impose au programme l'utilisant. La règle est que l'offset des fichiers ainsi que les adresses des tampons du programme doivent être alignées sur la taille des tampons, qui doit être elle-même une puissance de deux.

Ce n'est pas un souci dans le cadre de cet article mais les applications sophistiquées reposant sur O_DIRECT doivent gérer et aligner elles-même leurs tampons, en tenant compte de la mémoire disponible, du *swap*, des besoins des autres applications...

La réalisation du programme suivant soulève d'autres points épineux. D'abord, le disque est plus grand que la capacité d'adressage du off_t par défaut, qui est un long int. Il faut donc passer en mode 64 bits avec quelques #defines.

Ensuite, la déclaration de O_DIRECT n'est pas disponible dans les fichiers d'en-tête de la glibc, il faut accéder à la version fournie par le noyau.

Enfin, l'alignement et la taille du bloc lu posent plusieurs problèmes. En accédant au disque dur dev/hda, la taille est de 1024 octets au minimum (pour une raison que j'ignore).

Comme j'ai créé le programme avec l'idée préconçue qu'il était possible de lire un secteur de 512 octets à la fois, il a fallu du temps avant de comprendre pourquoi la lecture renvoyait systématiquement une erreur « Invalid argument ». L'utilisation de O_DIRECT est souvent très délicate et je ne suis pas le seul à avoir rencontré des problèmes au début.

Le souci majeur est de garder les adresses (du tampon et du fichier) alignées sur la taille du tampon. Pour pouvoir tester plusieurs tailles, le programme alloue dynamiquement la mémoire du tampon, ce qui impose des calculs et des vérifications.

En conséquence, le programme est simple (il repose sur seulement trois appels systèmes, surlignés dans le code) mais la vérification de chaque résultat allonge le code source.

```
001 /* fichier read_o_direct.c
002  créé par Yann GUIDON (whygee@f-cpu.org)
003  version Sun Oct 31 23:00:01 CET 2004
004
005  fonction :
006    Lit un secteur d'un périphérique spécifié
007    à l'offset indiqué. L'option "O_DIRECT"
008    lors de l'ouverture du périphérique diminue
009    le temps d'exécution lorsque le secteur
010    est défectueux, en empêchant les tentatives
011    répétitives de la part du kernel.
012
013  compilation :
014    gcc -o read_o_direct read_o_direct.c
015
016  invocation :
017    read_o_direct nom_périphérique numero_de_secteur
018      (option : taille du bloc lu)
019
020  valeur de retour :
021    succès si la lecture s'est bien passée.
022 */
023
024 /* les disques font plus de 2G aujourd'hui : */
025 #define _FILE_OFFSET_BITS 64
026 #define _LARGEFILE_SOURCE
027 #define _LARGEFILE64_SOURCE
028 /* merci à Chris pour les options 64 bits */
029
030 #include <asm/types.h>
031 #include <sys/types.h>
032 #include <sys/stat.h>
033 #include <linux/types.h>
034 #include <asm/fcntl.h> /* fcntl.h n'a pas O_DIRECT */
035 #include <errno.h>
036 #include <stdio.h>
037 #include <stdlib.h>
038 #include <unistd.h>
039
040 int main (int argc, char *argv[]) {
041   off_t secteur, resultat, buffsize;
042   int fd, t;
043   void *p;
044   char *buffer;
045
046   if (argc < 2+1) {
047     printf("Erreur d'argument\n\
048  Lancer avec nom_periph numero_secteur\
049  (option : taille du bloc)\n");
050     exit(EXIT_FAILURE);
051   }
052
053   secteur=strtoll(argv[2], NULL,10);
054   if (secteur < 0) {
055     printf("Erreur d'argument : offset négatif ?\n");
056     exit(EXIT_FAILURE);
057   }
058
059   if (argc > 3) {
060     buffsize=strtoll(argv[3], NULL,10);
```

```
061     if ((buffsize < 0)       /* teste si puissance de 2 */
062         || ( ((buffsize-1) & (~buffsize)) != (buffsize-1)
)) {
063       printf("Erreur d'argument : taille du bloc invalide
?\n");
064       exit(EXIT_FAILURE);
065     }
066   }
067   else
068     buffsize = 1024; /* valeur sûre */
069   printf("Taille du bloc : %lld\n", buffsize);
070
071   fd = open(argv[1], O_RDONLY|O_DIRECT|O_LARGEFILE);
072   if (fd == -1) {
073     perror("Erreur à l'ouverture ");
074     exit(EXIT_FAILURE);
075   }
076
077   /* calcule l'adresse du secteur en octets
078   (arrondi par défaut pour aligner sur les blocs) */
079   secteur = (secteur << 9) & (-buffsize) ;
080   printf("adresse de départ : %lld octets\n", secteur);
081
082   resultat = lseek(fd, secteur, SEEK_SET);
083   if (resultat != secteur) {
084     perror("Erreur de lseek() ");
085     exit(EXIT_FAILURE);
086   }
087
088   buffer=malloc(buffsize*2);
089   if (buffer == NULL) {
090     perror("Erreur de malloc() ");
091     exit(EXIT_FAILURE);
092   }
093
094   /* aligne le buffer sur une frontière naturelle */
095   p = (void *) ((((long)buffer)+(long)buffsize)
096                                 & (long)(-buffsize));
097   t = read(fd, p, buffsize);
098   if (t != buffsize) {
099     perror("Erreur de lecture ");
100     exit(EXIT_FAILURE);
101   }
102
103   exit(EXIT_SUCCESS);
104 }
```

2

Un peu d'aspirine

Ce programme a été développé sur un autre ordinateur et transféré sur la clé USB. L'exécution sur la machine cible n'a pas posé de souci. La taille par défaut de 1024 octets correspond aussi à la cible, mais la possibilité de changer cette taille permettra de remplacer `badblocks` sur des parties de cylindres. Contrairement aux scripts précédents, la valeur de retour de `read(2)` retourne une valeur souvent cohérente avec l'état du secteur et elle peut être utilisée directement. Un faux positif (tel que lors des essais avec `badblocks`) ne peut pas être complètement écarté. On doit donc conserver la fonction de chronométrage.

Le plus important est que le temps perdu est réduit à moins de quarante secondes lorsqu'un secteur défectueux est accédé. En fait, la durée de lecture d'un secteur défectueux est d'environ trente secondes, mais ensuite, le disque dur reste inaccessible durant six secondes. Il est évident que cela peut influencer l'algorithme de balayage du disque, surtout si le chronométrage est utilisé.

La solution est simple : il suffit de faire une requête sur un secteur connu comme valide. Cette requête (bloquante) se terminera lorsque le disque sera de nouveau prêt et le programme pourra continuer normalement. Il est probable que d'autres types de disque se comportent de manière différente. Il faudra donc adapter le script.

Tout ça pour ça ?

Bien content que le programme fonctionne finalement, et presque comme il aurait dû depuis le début, je tombe par hasard sur le *KernelTraffic n° 261* datant d'avril 2004.

Andy Isaacson y soulève les problèmes qui viennent d'être résolus. Seulement cela ne débouche pas sur un petit *hack* comme ici mais sur un *patch* pour l'utilitaire `dd`. Le patch a été intégré dans `coreutils` mais, à l'heure de la rédaction de l'article, la dernière version disponible dans les miroirs est la 5.2.1, antérieure d'un mois au patch.

Vérifiez donc la fraîcheur de votre distribution, et si vous avez une version récente, vous pourrez éviter de recopier le programme ci-dessus, pour le remplacer par `dd if=$dev of=/dev/null -iflags=direct count=1 bs=1024`.

Le petit programme est toutefois utile pour comprendre à quel point l'option `O_DIRECT` est sensible aux conditions d'utilisation, au cas où vous voudriez jouer avec avant de l'utiliser dans vos programmes.

On y arrive

Le script est en place et semble se comporter presque comme prévu, au prix de nombreuses modifications et de longues vérifications.

```
001 # fichier frag.sh, créé le 15 octobre 2004
002 #  version du 5 novembre 2004
003 # Ce script permet de scanner un disque "@$¤£*µ" de 20G
004 # - l'OS est le CDROM d'installation de Slackware 10.0
sur /dev/hdb
005 # - clé USB montée sur /d avec /d/frag.sh et les logs
006 # - bash est absent sur Slack/installation, je l'ai mis
sur la clé USB
007 # - disque sur /dev/hda
008 #
009 # taille du disque : 20003880960 octets
010 #    = 39070080 secteurs ( < à 4 milliards donc 32bits ok)
011 #    = 38760 cylindres de 16*63*512 octets (géométrie
physique)
012 # chaque cylindre fait 1008 secteurs, soit 516096 octets
013 #
014 # paramètre d'appel optionnel : numéro du premier secteur
à scanner
015 #  (pour pouvoir reprendre après une erreur ou un arrêt)
016
017 step=1008
018 step1=$((step / 4))
019 step2=$((step / 2))
020 step3=$((step1 + step2))
021 cyl=38760
022 dev=/dev/hda
023 log=/d/log/log20G
```

1

```
024 bb=/d/log/badblocks.out
025
026 # initialisation
027 if [ "$1"  ]
028 then
029   i=$1
030 else
031   i=0 # compteur de cylindres
032 fi
033
034 hdparm -d 0 -m 0 -a 0 -A 0 $dev
035
036 echo ---------------------------  >> $log
037 date >> $log
038 echo ---------------------------  >> $log
039 echo debut=$i
040 echo debut=$i >> $log
041
042 # teste un secteur donné
043 function chrono () {
044 # argument : n° du secteur
045   SECONDS=0   # remet le compteur à zéro
046   if /d/read_o_direct $dev $1
047   then
048     if (( $SECONDS > 3 ))
049     then
050       echo temps = $SECONDS s : trop long !
051       echo secteur $1 KO >> $log
052       return 1
053     fi
054   else
055     echo ' --- la commande a échoué. ---'
056     echo secteur $1 KO >> $log
057     return 1
058   fi
059
060   return 0
061 }
062
063 # teste plusieurs secteurs régulièrement espacés
064 function test_secteurs () {
065   k=5 # $j+0 a déjà été testé
066   while (( $k < $step ))
067   do
068     if ! chrono $(( $j + $k ))
069     then
070       return 1
071     fi
072     k=$(( k + 10 ))
073   done
074   return 0
075 }
076
077 function test_cylindre () {
078   rm -f $bb
079
080   echo "cylindre $i :"
081
082   SECONDS=0  # remet le compteur à zéro
083   if ! badblocks -o $bb -s -v -b 512 $dev \
084      $((($j)+ step -1)) $j
085   then
086     cat $bb >> $log
087     return 1
088   fi
089
090   if [ -s $bb ]
091   then
092     echo "cylindre $i HS :"
093     return 1
094   fi
095
096   if (( $SECONDS > 5 ))
097   then
098     echo temps de badblocks = $SECONDS s : trop long !
099     echo -n "cylindre $i lent $SECONDS s " >> $log
100     return 1
101   fi
102
103   return 0
104 }
105
106 # Début de la partie principale
107 function principale () {
108   while (( $i < $cyl ))
109   do
110     j=$((i * step))
111     echo ----------------------------
112     echo ---- cylindre $i ----
113     echo ----------------------------
114
115     if ! /d/read_o_direct $dev 0
```

2

```
116     then
117       echo "Disque dur inaccessible" >> $log
118       echo "Disque dur inaccessible"
119       return 1
120     fi
121
122     sync
123
124     etat_courant=' raté'
125
126     # 'if's embriqués :
127     chrono       $j               &&
128      chrono      $((j + step1   )) &&
129       chrono     $((j + step2   )) &&
130        chrono    $((j + step3   )) &&
131         chrono   $((j + step - 2)) &&
132          test_secteurs            &&
133           test_cylindre           &&
134            etat_courant='ok'
135
136     echo cylindre = $etat_courant
137     echo "cyl $i = $etat_courant" >> $log
138
139     i=$((i + 1))
140   done
141
142   echo fini !
143 }
144
145 principale && echo "Disque dur complétement scanné !"
>> $log
```

3

La vérification totale du disque a duré 36 heures.

Une fois les erreurs de programmation corrigées, il est apparu que de nombreux cylindres passaient correctement les tests mais `badblocks` était très lent. Lire 500KO en 14 secondes représente un débit de 35KO/s, le balayage complet du disque est donc très très long.

Ces cylindres sont souvent entourés d'autres cylindres défectueux, ce qui signifie que la zone est endommagée et la reconstruction des données risque de ne plus être possible plus tard, il faut aussi noter les secteurs voisins comme invalides.

La vérification de chaque cylindre est effectuée en trois étapes afin de perdre le moins de temps possible.

- D'abord, cinq secteurs sont lus avec `read_o_direct`. Ce sont les premiers et derniers secteurs, le secteur du milieu, et les deux secteurs entre ceux-ci. Cela permet de détecter un cylindre très endommagé et de ne pas s'y attarder.

- Une vérification moins grossière est effectuée en lançant `read_o_direct` à l'intérieur d'une boucle au cas où des zones défectueuses plus petites existent.

- Enfin, `badblocks` est lancé en lecture seule pour s'assurer que le cylindre est (au moins) lisible intégralement. Il n'est pas rare de trouver de très nombreux secteurs défectueux encore à ce stade.

Le but de ce script est de trouver des zones valides contiguës. Il n'est alors pas encore nécessaire de balayer chaque cylindre avec `badblocks` en mode écriture (ce sera effectué ensuite après le partitionnement). Avant cela, il faut éliminer les zones qui ne sont pas parfaitement utilisables.

Au rayon des petites astuces, en voici une qui remplit plusieurs fonctions :

```
115     if ! /d/read_o_direct $dev 0
116     then
118       echo "Disque dur inaccessible"
119       return 1
120     fi
```

Ce code est situé au début de la boucle, avant les premiers tests. Cet accès au disque à un endroit connu pour être valide a d'abord un effet temporisateur. Il permet d'attendre que le disque soit de nouveau accessible.

En premier lieu, si l'ordinateur était inactif avant de lancer le code, cet accès va provoquer la rotation des plateaux et éviter que le premier test n'échoue à cause du délai.

De la même manière, dans la boucle, cet accès succède à un test précédent. Si ce dernier a accédé un secteur difficilement lisible (entraînant le crépitement du disque et le blocage du système), l'accès à la zone valide permet d'attendre que le disque ait fini de crépiter (les six secondes de latence après une erreur).

Enfin, si cet accès échoue, c'est que la trop grande quantité de secteurs défectueux a bloqué le disque et le programme ne peut pas continuer plus loin.

Un peu de rétrospection

L'ordinateur passe beaucoup de temps à le perdre. D'abord, `sync` est indispensable à chaque itération car si l'ordinateur plante, on veut savoir où. Mais cet appel consomme une fraction de seconde pour chaque demi-mégaoctet ce qui ralentit le balayage des zones valides.

En plus, il ne m'est venu à l'idée qu'à la fin du test que le `sync` systématique n'était probablement pas une bonne idée pour la clé USB. Je ne sais pas si la mémoire Flash qu'elle contient est capable de supporter encore beaucoup de cycles d'écriture après les 38000 cycles déjà effectués.

L'autre option était de monter la clé avec `mount -o sync /dev/sda /d` mais alors chaque petit ajout au `log` (comme les messages d'erreurs intermédiaires) aurait ralenti l'ordinateur.

Il va falloir trouver une meilleure technique, moins pénalisante et moins stressante, d'assurer la journalisation des vérifications.

Un autre ralentissement a pour cause le lancement de `badblocks` sur des zones « lentes ». Avec une granularité de 516096 octets, il n'est pas possible de ne balayer qu'une partie car la précision du chronométrage est trop faible.

Enfin, l'exécution d'un script `bash` et de programmes externes (même si ce sont toujours les mêmes) est un surcoût dont il faudrait se passer.

L'évolution logique du programme serait de le recoder en C, permettant des chronométrages fins et l'intégration de fonctions sophistiquées (sans appel système pour les lancer) mais la console « simplifiée » de Slackware 10.0 n'offre pas de compilateur.

De toute façon, le balayage du disque est d'une grande lenteur inhérente.

Dépouillement

Dès le début du développement du script, il était question que celui-ci se charge de définir les zones valides en détectant les cylindres valides contigus. La syntaxe de `bash` et le comportement du disque n'ont pas rendu cela possible.

Il faut donc examiner le fichier de `log` à la main. Heureusement, les outils UNIX classiques aident à trier les données. On peut déjà connaître les statistiques du disque avec `grep` et faire un peu de nettoyage, comme enlever tous les numéros de secteurs défectueux qui prennent de la place.

Finalement, malgré la grande quantité de cylindres hors service éparpillés au début du disque, il fut facile de trouver deux grandes zones utilisables, ainsi que plusieurs zones d'environ 50MO.

Ces dernières seront laissées de côté, d'une part parce qu'elles sont trop petites pour être utiles, d'autre part parce que le risque de dégradation de la surface au voisinage des zones mortes est grand.

C'est aussi pour cette dernière raison que j'ai arrondi les adresses des cylindres

des zones trouvées comme valides, à l'exception de la première zone où les premiers tests ont déjà provoqué la réallocation des secteurs.

Le script de balayage n'a fait que lire les secteurs. Les échecs ont été mémorisés par le contrôleur du disque. Si on écrit sur un secteur qui a été préalablement marqué comme défectueux, le contrôleur écrira en fait ces données sur un autre secteur de la « zone de réserve ».

Si on avait lancé `badblocks -w`, tous les secteurs défectueux auraient été écrits et il ne resterait alors plus de place dans la « zone de réserve » pour les petites erreurs qui seraient survenues plus tard dans les zones réellement utilisables.

Le `log` indique que presque 2GO de secteurs sont endommagés ou inutilisables. Parfois, il n'y a qu'un seul secteur défectueux sur un cylindre, parfois au moins une centaine...

Verdict

Les trois zones sont :

```
zone 1 : secteurs     0 - 89
zone 2 : secteurs  1450 - 21200
zone 3 : secteurs 22700 - 38760
```

`smartudm` retourne ces chiffres du contrôleur :

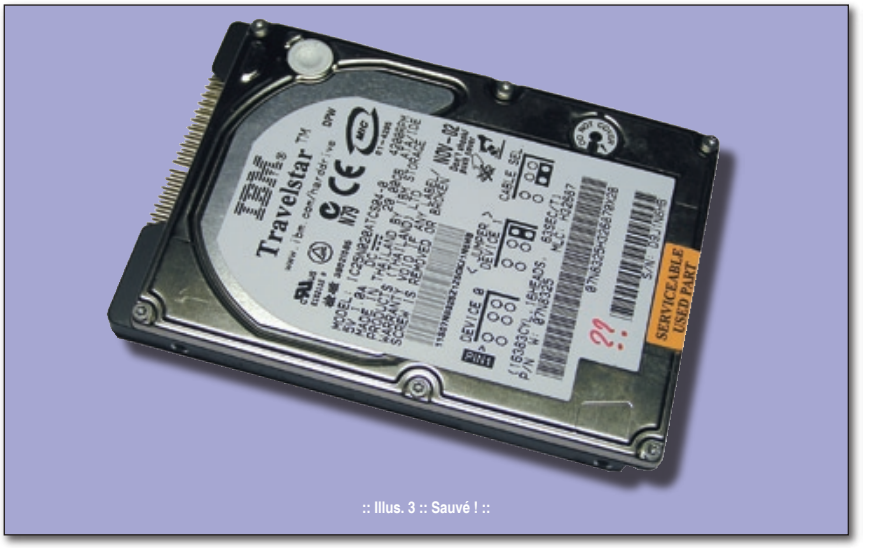
```
Reallocation Event Count : 446
Current Pending Sectors  : 1106
```

Le verdict définitif ne tombe pas tout de suite car `badblocks -w` réserve encore des surprises, puisque seul un test en lecture a été effectué jusque maintenant.

La première zone, petite et dédiée au démarrage, demande plusieurs passes car le balayage « gratte » encore. Au bout de trois passes, il n'y a plus de bruit. Ce sera donc la partition de boot.

La troisième zone de 8GO ne pose aucun souci. Deux longues passes permettent de s'en convaincre, mais tendre l'oreille pendant de longues heures, en faisant autre chose, demande beaucoup de concentration...

La deuxième zone pose plus de problèmes. J'ai dû en repousser le début de presque trois Gigaoctets avant de trouver une zone qui ne crépite pas. Mais même ensuite, il y a des petites zones « fatiguées » qui n'apparaissent qu'au bout de plusieurs balayages.



:: Illus. 3 :: Sauvé ! ::

Il est évident que la sensibilité du chronométrage est très laxiste et il faudrait une précision de l'ordre de la milliseconde pour détecter individuellement les secteurs posant problème, pas seulement les illisibles mais aussi ceux que le contrôleur du disque arrive à lire après plusieurs essais.

J'aurais voulu écrire un programme pour résoudre ce problème mais il ne reste plus assez de temps ni de place dans cet article. Un logiciel spécialisé ou une modification de `badblocks` pourrait donner une liste de secteurs à éviter pour `mke2fs`

J'avais espéré que de nombreuses passes de `badblocks -w` viendraient à bout des crépitements résiduels de la deuxième zone. En réalité, certaines passes ne posaient aucun problème, mais la passe suivante retournait de nombreuses erreurs.

Devant le caractère imprévisible de cette zone, j'ai préféré l'abandonner. Il reste donc 8 Gigaoctets d'espace utilisable sur le disque, assez pour installer une *slackware*. Les performances ne seront pas fulgurantes car la grande zone utilisable est à la fin du disque, dans la partie la plus lente, comme nous allons le voir bientôt. Il ne faudra pas non plus s'attendre à une stabilité sans faille en raison de l'instabilité de la surface, mais cela suffit pour bricoler.

Un bon point tout de même : nous avons vu que le noyau avait changé la géométrie logique du disque en cours de route. Cela n'est plus un problème une fois que la table des partitions est écrite ! Il faut juste lancer `fdisk` après avoir modifié `/proc/ide/hda/settings` comme décrit précédemment.

Acte IV: on va finir par s'en sortir

Encore un peu de paranoïa

Les zones valides sont délimitées mais il ne faudra pas relâcher l'attention durant toute la vie active du disque. Les multiples passes de balayage montrent que des secteurs défectueux apparaissent au fur et à mesure.

Puisque la correction automatique des secteurs ne fonctionnera peut-être plus correctement à cause de la saturation de l'espace de secours, il faudra tester et « corriger » manuellement les partitions, donc lancer `badblocks -n` (test en lecture seule) de temps en temps sur chaque partition.

Si une erreur supplémentaire est trouvée, il y a deux solutions :

- re-balayer le disque avec les programmes pour localiser la nouvelle zone défectueuse, puis découper la partion endommagée afin de ne pas utiliser les secteurs défectueux.
- si les erreurs sont en faible nombre, relancer `mke2fs -c` pour que le système de fichiers mette les secteurs défectueux à l'écart.

Evidemment, les données présentes sur la partition endommagée doivent être recopiées (si elles ne sont pas perdues)

sur une autre partition ou un autre disque, avant d'effectuer les opérations de formatage. Afin de faciliter la recopie d'une partition à une autre, il faudrait créer plusieurs partitions de taille similaire. Une partition pourra même servir d'espace temporaire dédié.

Un peu de géométrie

Le disque dur étant maintenant « sain », ou du moins apparemment sous contrôle, la suite consiste à partitionner le disque et à formater chaque partition.

Si le disque contient seulement des données, la situation est simple. Il peut être divisé en plusieurs parties de tailles plus ou moins égales comme indiqué plus haut, afin de laisser un degré de liberté au cas où une zone de secteurs défectueux devait apparaître. Plus les partitions sont petites, moins il y a de données à déplacer.

Pour ce qui est d'installer un système *unixidé*, un paramètre entre en jeu : la partition de *swap*. Où la mettre ? Au début du disque, où le transfert est plus rapide, ou à la fin ?

Nous allons voir qu'il est préférable, si le swap est très utilisé, de placer la partition environ à deux tiers du disque.

La justification est que, dans le pire des cas, la tête aura le moins de trajet à faire, puisque les secteurs à deux tiers de la capacité sont environ à mi-chemin (pour la tête) entre le début et la fin du disque.

S'il faut compiler un noyau ou KDE/GNOME (par exemple), il faudra aller lire et écrire des fichiers un peu partout sur le disque, et sans ordre apparent. Si le swap est au milieu, cela divise par deux les temps de positionnement par rapport à une partition en position extrême.

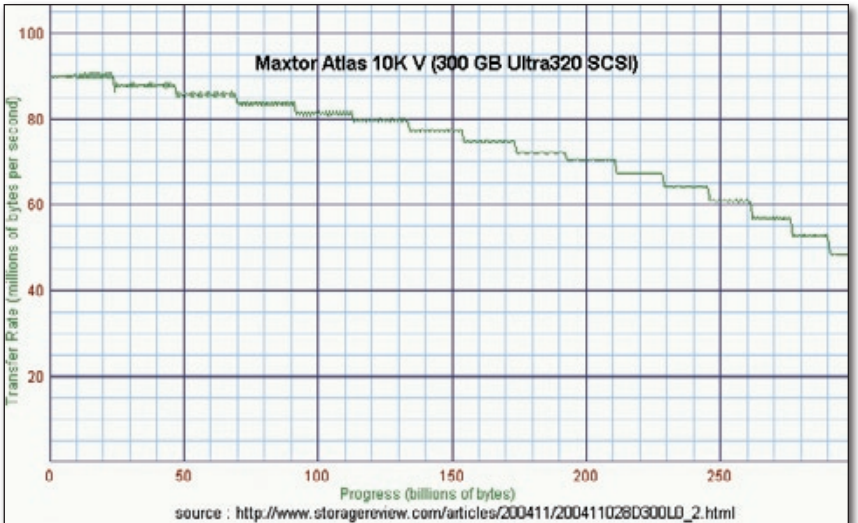
Cette approche est utilisée par exemple par les disquettes, dont la table de partition est située dans le cylindre du milieu. Cela réduit d'un facteur deux le temps maximal de recherche des fichiers, ce qui n'est pas négligeable en raison de la lenteur de ce type de périphérique.

Mais pourquoi installer le swap aux deux tiers au lieu de la moitié du disque ? En fait, contrairement à une disquette où le nombre de secteurs par cylindre est fixe, les disques durs actuels exploitent au maximum la surface en mettant moins de secteurs vers le centre du disque qu'à la périphérie.

Donc le numéro d'un secteur n'est pas linéairement proportionnel à sa position sur le disque. Comme la densité en secteurs est plus faible à la fin (proche du centre), le secteur correspondant à la moitié « géométrique » du disque ne correspondra pas au secteur du milieu, mais un peu plus.

Le graphique suivant montre l'influence de la géométrie d'un disque sur sa vitesse de lecture :

:: Illus. 4 :: La vitesse de défilement sous la tête de lecture, donc la densité des cylindres, est environ 2,8 fois plus grande à l'extérieur qu'à l'intérieur de ce disque. ::

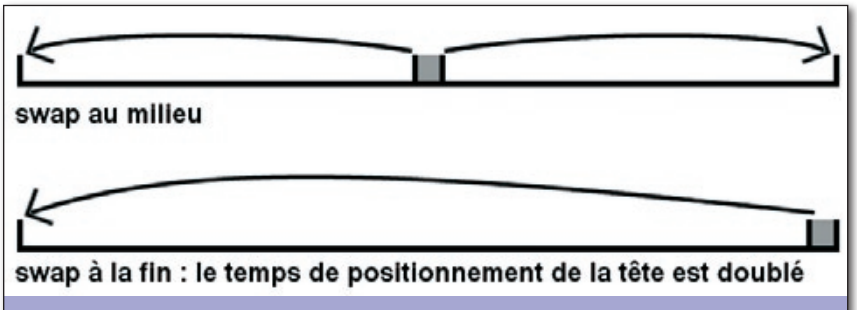


:: Illus. 5 :: Débit réel d'un disque en fonction de la position de la tête. ::

Ce graphique révèle plusieurs choses :

- Le débit maximum soutenu est de 89,5MB/s : c'est l'un des disques durs les plus rapides à l'heure où cet article est écrit, mais malgré l'utilisation d'une interface SCSI Ultra320, il ne dépasse pas le débit maximal théorique d'une interface ATA100.
- Il y a 16 paliers, correspondant à 16 zones du disque avec des densités de bits différentes. Le débit varie presque d'un facteur 2 !
- La courbe descend ce qui veut dire que le début du disque, vu par l'ordinateur, est à l'extérieur, comme pour les disques en vinyle et contrairement aux CD-ROM !
- Cela veut aussi dire que le résultat de `hdparm -t -T`, qui mesure la vitesse de lecture au « début » du disque, donne bien un maximum.

En ce qui concerne le disque dur pour portable qui vient d'être récupéré, le débit de 20MO/s du début chute à 15MO/s au milieu et à 10MO/s à la fin.



:: Illus. 3 :: Sauvé ! ::

La précision du partitionnement n'est pas très importante puisque le résultat dépendra aussi du placement des fichiers sur le disque. Cela permet par contre de s'assurer que dans le pire des cas, les temps d'accès seront plus réduits.

Encore un rappel : le meilleur moyen de réduire l'impact du swap n'est pas d'utiliser un *ramdisk* mais d'avoir beaucoup de mémoire ;-)

Diviser pour régner

Voici par exemple la table des partitions sur mon laptop de travail. Il *swappe* rarement mais si cela arrive, la partition d'échange se situe entre la partition système et les gros fourre-tout, réduisant le trajet de la tête de lecture.

```
root:~# fdisk -l /dev/hda

Disk /dev/hda: 255 heads, 63 sectors, 3648 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1         7     56196    6  FAT16
(partition pour booter)

/dev/hda2             8      3648  29246332+   f  Win95 Ext'd
(LBA)

/dev/hda5             8       390   3076416   83  Linux
(racine du système)

/dev/hda6           391       773   3076416   83  Linux
(deuxième racine pour un autre linux)

/dev/hda7           774      1156   3076416   83  Linux
(troisième racine pour un autre système)

/dev/hda8          1157      1666   4096543+  83  Linux
( /common, données communes aux systèmes)

/dev/hda9          1667      1743    618471   82  Linux swap
(swap vers le milieu)

/dev/hda10         1744      2508   6144831   83  Linux
( /data1 pour mettre plein de fichiers

/dev/hda11         2509      3648   9157018+  83  Linux
( /data2 pour mettre le reste)
```

Si vous avez lu l'article en entier, je suppose alors que vous savez déjà vous servir de `fdisk` ou d'autres programmes équivalents de votre choix. Une fois les partitions créées, ne pas oublier de formater (`mke2fs -j` et `mkswap` par exemple).

N'oublions pas non plus qu'un système de fichiers journalisé permet de réduire la casse en cas d'arrêt brutal. L'intégrité des fichiers est mieux garantie et cela évite souvent un long `fsck` lors du redémarrage.

Pratiquement tous les systèmes récents sont journalisés, comme Reiserfs, JFS, Ext3, mais pas Ext2.

Dernier acte : on va conclure

Nous sommes à la fin d'un article très varié avec de la géométrie, de l'UNIX, du Linux, du Zen, un soupçon de social *engineering*, du *tuning*, du C à la limite du POSIX, du *scriptage*, des soins capillaires...

Au passage, la puissance et la flexibilité de GNU/Linux ont été mises en valeur et sont venues à bout d'un problème grave. Cela a rendu dignité et utilité à un disque qui aurait été jeté par son propre fabricant. Vu le prix d'un disque dur neuf pour portable, les efforts ont payé.

Je tiens à souligner que vos propres expériences vont dépendre de nombreux facteurs que je n'ai pas pu aborder, par ignorance ou manque de place. Cet article ne peut prétendre être omniscient ou même apporter de solution miracle à tous les problèmes que vous pourrez rencontrer. Les techniques utilisées par les disques durs sont tellement sophistiquées et variées que les outils de base ne sont pas suffisants.

Tout cela pour insister sur l'aspect préventif de la pérennisation des données. Des outils existent pour récupérer les données après incident mais il est souvent déjà trop tard, surtout si le support est physiquement endommagé. A l'acquisition d'un nouveau support, s'il n'y avait que deux choses à retenir, ce serait :

- 1) Lancer immédiatement les utilitaires de diagnostic du fabricant pour vérifier la santé de l'unité.
- 2) Puis lancer `badblocks -w` sur l'unité, plusieurs fois si possible, et prêter l'oreille aux crépitements qui signalent des zones « difficiles » à surveiller.

L'entretien consistera, en plus de l'utilisation de `smartd`, à lancer régulièrement `badblocks -n` (en mode non-destructif) sur toutes les partitions, bien que ces modes de vérification ne permettent pas de détecter toutes les anomalies. Par exemple, `smartd` seul ne préviendra pas de la dégradation de la surface dans des zones inutilisées. Des applications installées par défaut qui ne servent à rien ou de vieux fichiers qui traînent pourraient mourir sans prévenir.

J'ai décrit un cas extrême mais j'espère avoir provoqué une prise de conscience : la Loi de Murphy guette les utilisateurs trop confiants. La plupart des disques durs sont raisonnablement fiables mais j'ai noirci le tableau pour que cette confiance ne vous aveugle pas et qu'il reste une parcelle de doute dans votre esprit, pour que vous pensiez que votre disque dur, celui qui ronronne dans votre ordinateur en ce moment, est « tombé en marche » une fois de plus et que cela peut s'arrêter d'un moment à l'autre. La prochaine fois que vous achèterez un disque dur, vous saurez comment vérifier son intégrité, et s'il lâche, ou s'il s'agit d'un disque dur d'occasion, comment utiliser au maximum sa capacité restante (et peut-être en profiter pour baisser le prix).

Pour terminer, merci à **StorageReview.com** pour la permission d'utiliser une de leurs illustrations et aux noctamoules pour leur aide :-)

Références

- SMART Monitoring Tools : http://smartmontools.sourceforge.net
- [1] http://smartmontools.sourceforge.net/BadblocksHowTo.txt
- D'autres conseils : http://forum.hardware.fr/hardwarefr/Hardware/sujet-605153-1.htm
- The Ultimate Boot CDrom : http://ultimatebootcd.com
- IBM/HGST Drive Fitness Tool : http://www.hgst.com/downloads/DFT32-V340.EXE
- Maxtor MaxDiag : http://www.maxtor.com/en/support/downloads/files/powermax.exe
- Seagate SeaTools : http://www.seagate.com/support/npf/seatools/seatoold.exe
- Western Digital Datalifeguard : http://support.wdc.com/download/dlg/dlginstall_10_0.exe
- Humour : http://www.forum-auto.com/sqlforum/section7/sujet46623.htm