- „noop“ - wie der Name schon sagt, scheduled der gar nichts. Simples FIFO-Queuing für alle I/O Requests egal woher sie stammen. Ist gedacht für wirklich intelligente Hardware, die ihr eigenes internes Scheduling macht, und wo jedes Scheduling durchs OS kontraproduktiv wäre. (Etwa SANs, könnte ich mir vorstellen.) Update: Angeblich ist dies auch der beste Scheduler für Flash-Medien.
- „deadline“: I/O-Requests werden nach der Blocknummer sortiert in eine Warteschlange eingeordnet unabhängig woher sie stammen. Diese Queue wird dann zyklisch von vorn nach hinten abgearbeitet. Damit I/O-Requests nicht unmäßig warten müssen falls die Schlange zu lange wird, werden die I/O-Requests überdies in separate FIFOs für Read und Write-Requests eingeordnet, nach Alter sortiert. Jedem Request wird dabei eine Maximaldauer zur Erledigung zugewiesen. Solange diese Maximaldauern nicht überschritten werden werden die I/Os wie zuvor beschrieben in der Sortierreihenfolge der Blocknummern durchgeführt. Sobald die Maximaldauern aber überschritten werden, werden nur mehr abwechselnd die ältesten Read- und Write-Requests aus den FIFOs bedient, bis sich die Situation wieder gebessert hat - dann wird wieder das zyklischen Schedulen in Blockreihenfolge angeworfen.
- „anticipatory“: Tut dasselbe wie „deadline“, wartet aber nach jedem I/O-Vorgang (ich hoffe nur bei Schreibvorgängen aus den Deadline-FIFOs) einige Millisekunden ob vielleicht ein Nachfolge-Request daher kommt. Dadurch werden zahlreiche sequenzielle I/Os besser unterstützt, welche an verschiedenen Stellen der Disk gleichzeitig erfolgen. Sprich, wenn verschiedene Prozesse zur selben Zeit verschiedene Dateien bearbeiten. Der „deadline“ geht dabei nämlich ziemlich ein und „seekt“ sich blöd.
- „cfq“: Der „Completely Fair Queueing“ Scheduler arbeitet völlig anders. Er scheduled Zeitscheiben, in denen nur die I/O eines bestimmten Prozesses auf die Disk erfolgt. Die Größe der Zeitscheiben ist von Statistiken und der Prozesspriorität abhängig, und ausserdem wird innerhalb eines Prozesses noch zwischen synchronen und asynchronen Requests unterschieden. Aber zwischen den Prozessen gibt es ein Round Robin, daher kommt jeder Prozess nach einer relativ kurzen Zeit wieder dran, und keiner „verhungert“ auch wenn sehr viel I/O durchzuführen ist.

Der cfq ist der komplexeste der I/O Scheduler, reagiert aber am schnellsten. Daher ist es für den typischen Desktop-Betrieb normalerweise der am besten geeignete I/O Scheduler. In vielen Linux-Distris wird er daher auch als Default-Scheduler eingesetzt.

Allerdings scheduled der cfq die Requests nicht optimal aus der Sicht der Arbeit welche die Disk zu erledigen hat. Hier ist der anticipatory normalerweise am effizientesten - insbesondere wenn Batch-Jobs laufen die ihrerseits jede Menge sequenzielle Dateien (gleichzeitig) bearbeiten. Auch zum Ansehen von Videos etc. ist er wohl der geeignetste.

Der deadline wiederum glänzt bei völligen Random-Zugriffen, wie sie vor allem bei Datenbanken oft vorkommen. Hier sorgt er dafür, dass die Seeks minimiert werden welche für das Durchführen der Random-Zugriffe erforderlich sind. Zwar ist der anticipatory dem deadline sehr ähnlich, aber durch die kleinen Wartepausen die er einlegt um „Sequenzialität zu erkennen“ (die bei Datenbankzugriffen nicht vorkommt), vergeudet das anticipatory Zeit welche der deadline nicht vergeudet.

Wenn Datenbankzugriffe aber nicht andauernd erfolgen, kann der anticipatory doch wieder besser sein: Bei Datenbankzugriffen zwar etwas langsamer, kann er aber zwischendurch bei sequenziellen Zugriffen wieder Zeit und Seeks sparen.

Wenn neben den Datenbanken und Batch-Jobs aber auch noch „normal“ gearbeitet werden soll, empfiehlt sich wieder der cfq: Durch seine Zeitscheiben werden auch sequenzielle Jobs - zumindest

innerhalb der Zeitscheibe - halbwegs effizient abgearbeitet, durch seine verschieden langen Zeitscheiben kann er aber auch Datenbanken ausreichend effizient bedienen obwohl nicht ganz so gut wie der deadline. Vor allem aber verhungern während dessen keine interaktiven Benutzerprozesse.

Ich werde aus diesen Erkenntnissen die Konsequenz ziehen, den cfq als Default-Scheduler einzustellen.

Wenn ich aber fette Batch-Jobs laufen lasse, wie große emerge-Orgien wo der Compiler ständig sequeziell Source-Dateien liest und Object-Files erzeugt, werde ich temporär auf den anticipatory umschalten. Dasselbe gilt beim Movie-Ansehen, oder wenn sehr große Dateien möglichst schnell durch die Gegend kopiert werden sollen und mir Interaktivität währenddessen nicht so wichtig ist.

Wenn ich hingegen einen Rechner als dezidierten Datenbankserver unter hoher Last einsetze, ist hingegen der „deadline“ die beste Wahl. (cfq dürfte auch OK sein wenn die Last nicht ganz so hoch ist.)

Tja, soweit meine Erkenntnisse.

Hier noch wie man die Scheduler umschaltet (geht im laufenden Betrieb):

<pre>

1. ! /bin/sh

SCHEDULER=${1:-cfq} lsmod | grep „$SCHEDULER[_-]iosched“ > /dev/null 2>& 1 || {

```
modprobe "$SCHEDULER-iosched"
```

} for D in /sys/block/*; do

```
S="$D/queue/scheduler"
       test -e "$S" || continue
       echo "Assigning $SCHEDULER to $S."
       echo "$SCHEDULER" > "$S"
```

done

Aufruf: <pre> set_iosched cfq set_iosched # setzt ebenfalls cfq set_iosched noop set_iosched anticipatory set_iosched deadline

Oder für ein Blockdevice setzten: <pre>echo anticipatory > /sys/block/hdc/queue/scheduler