Der "Fair ShareScheduler

Einführung
Ein Hauptproblem, das jeder, der etwas verwaltet, zu lösen hat, ist das Verteilungsproblem: Wer bekommt wann wieviel? An einem Rechenzentrum gibt es allerlei Größen, auf die dies zutrifft, am meisten aber gilt es wohl für die CPU-Zeit. Um die CPU-Anforderungen vieler User zu befriedigen, wurden schon bald Programme entwickelt, die als Schiedsrichter auftreten, sogenannte Scheduler. Ein Scheduler reiht alle Anfragen in eine Warteschlange ein, ggf. auch in mehrere, falls einige User bereit sind, auf die höchste Dringlichkeitsstufe zu verzichten, und bedient alle der Reihe nach. Dabei bekommt jeder die CPU nur für eine bestimmte Zeit, die Zeitscheibe, die typisch 1/60 s beträgt, und muß sich dann wieder hinten anstellen. Die User allerdings sannen erfolgreich auf Wege, die Warteschlange zu umgehen. Eine einfache Methode ist, sich nicht einmal, sondern mehrmals anzustellen, d.h. mehrere Prozesse gleichzeitig zu starten. Die geplagten Rechenzentrumsbetreiber sannen auf Abhilfe und suchten ein Verfahren, jedem Benutzer einen gerechten Anteil an der CPU zu geben, und so entstand der Fair Share Scheduler.

Prinzip des Fair Share Schedulers
Der Fair Share Scheduler (FSS) wurde an der Universität Sidney mit doppelter Zielsetzung entwickelt: Zum einen sollte jedem User ein bestimmter Anteil an der Maschine zustehen, ohne daß ein Leerlauf entstehen kann, zum andern sollte das Verhalten der Maschine (unterschiedliche Antwortzeiten) für den Benutzer vorhersehbar sein.

Für das Verständnis des FSS sind für einen Benutzer zwei Begriffe wichtig:

1. der Anteil der Maschine, der einem User zusteht (i.f. "Share" genannt oder auch "intended Share"), eine Größe, die nur in Relation zu den Shares anderer User etwas aussagt,
2. die "Usage", ein Maß für den CPU-Verbrauch eines Users "in der letzten Zeit". Die Usage steigt beim Verbrauch von Resourcen (CPU), außerdem verringert sie sich für alle User mit einer (vom Systemmanager einstellbaren) Halbwertszeit.

Das Verhalten des Schedulers ist dann folgendermaßen: Bei (im Schnitt) gleichmäßiger Auslastung der Maschine durch alle User bekommt jeder den seinem Share entsprechenden Anteil, die Usage (oder genauer: die normierte Usage = Usage/Share) steigt gleichmäßig. Wenn einige User weniger als ihren Anteil verbrauchen, ist ihre Usage im Vergleich niedriger, sie bekommen beim nächsten Mal dann einen höheren Anteil. Wer mehr als seinen Anteil verbraucht hat, wird dies (bei gestiegenem Bedarf anderer User) am schlechteren Antwortverhalten des Systems merken. Da aber die Usage mit fester Halbwertszeit zerfällt, kann er durch Abwarten diesem Mißstand abhelfen.

Solange sich nur einige eingetragene Benutzer die Maschine teilen, bekommen sie also - wie früher - einen großen CPU-Anteil. Logged sich jetzt jedoch ein User ein, der selten arbeitet, bekommt er soviel CPU, wie er braucht, um kleine Aufgaben zu erfüllen. Damit kann also trotz

überlasteter Maschine ein gelegentlicher User noch seine Mail abschicken, die News lesen oder ein File edieren, ohne überhaupt von den "Power-Usern" gestört zu sein.

Durch geeignete Wahl der Antwortzeit hat der Systemmanager die Möglichkeit, das Userverhalten zu beeinflussen: Halbwertszeiten von wenigen Stunden bewirken ein Verteilen der Arbeiten über den Tag, dagegen wird eine Halbwertszeit von zwei Tagen die User zur Verteilung ihrer Arbeiten über die Woche anhalten.

Eine weitere Möglichkeit der Einflußnahme besteht im Verändern der "Kosten", d.h um wieviel die Usage bei bestimmtem Verbrauch zunimmt. So kann man z.B. das Arbeiten außerhalb der "Stoßzeiten" durch geringeres Zubucheschlagen auf dem Usage-Konto belohnen.

Funktionsweise des FSS
Der FSS benötigt zur Buchhaltung die folgenden Größen:

1. die Usage jedes Users (usage(U)),
2. eine Zwischensumme für den (CPU-)Verbrauch eines Users ($t_{charge}(U)$),
3. die Prozeßprioritäten aller Prozesse (prio(P)),
4. die Anzahl der Prozesse eines Users (#procs(U)).

Dabei ist zu beachten, daß unter UNIX ein großer Wert prio(P) bedeutet, daß man weniger CPU bekommt.

Die Arbeit des FSS geschieht nun auf drei Zeitskalen $t1 > t2 > t3$ (z.B. 4s, 1s, 1/60 s):

- Alle t1 Sekunden: Scheduling auf User-Ebene
  Für alle User wird die Usage entsprechend der Halbwertszeit verringert, die inzwischen angesammelten Kosten werden ihr zugerechnet und der Kosten-Zwischenzähler auf Null gesetzt.
  In Formeln:
  $$usage(U) := k1 \cdot usage(U) + t_{charge}(U) \quad (k1 < 1)$$
  $$t_{charge}(U) := 0$$
- Alle t2 Sekunden: Prozeßprioritäten verringern
  Die Prioritaten aller Prozesse werden verringert, d.h. sie wandern in der Schlange (aufgrund ihrer Wartezeit) nach vorne. Dabei wird ggf. noch ein nice-Wert berücksichtigt.
  In Formeln:
  $$prio(P) := prio(P) \cdot k2 \cdot (nice(P) + k3)$$
- Alle t3 Sekunden: Priorität des laufenden Prozesses vergrößern
  Der laufende Prozeß wird in der Schlange nach hinten gestellt, und zwar umso weiter, je höher die Usage und die Anzahl der Prozesse des zugehörigen Users sind, und umso weniger, je größer sein Share ist. Gleichzeitig wird ihm der bisherige (CPU-)Verbrauch auf sein Konto gebucht.
  In Formeln:
  $$\begin{aligned} prio(P) &:= prio(P) + \frac{usage(U) \cdot \#procs(U)}{shares(U)^2} \\ t_{charge}(U) &:= t_{charge}(U) + cost(CPU) \end{aligned}$$

Anmerkung: Dies ist eine vereinfachte Darstellung, weitere Details s.u. .

Der hierarchische Share
Geht es nur darum, die Maschine zwischen mehreren Usern aufzuteilen, ist das beschriebene Verfahren ausreichend. Probleme treten jedoch auf, wenn die User in Gruppen organisiert sind, zwischen denen die eigentliche Verteilung geschieht. Ein Beispiel möge dies verdeutlichen:

Wir betrachten zwei Gruppen g1 und g2, die jeweils 50% Anteil haben sollen. In jeder Gruppe seien zunächst zwei User u11, u12 und u21, u22, die jeweils einen Share von 100 haben. Damit bekommt jeder User 25%, die Gruppen also 50%, wie verlangt. Kommt nun in die Gruppe g2 ein weiterer User u23, der den gleichen Anteil haben soll wie u21 und u22, möchte man ihm zunächst einen Share von 100 zuteilen. Dann hat aber die Gruppe g2 insgesamt 300 gegen 200 Shares von g1, bekommt also nun 60%.

Die einzige Möglichkeit, die Balance der Gruppen zu gewährleisten, ist eine völlige Neuverteilung aller Shares, etwa so:

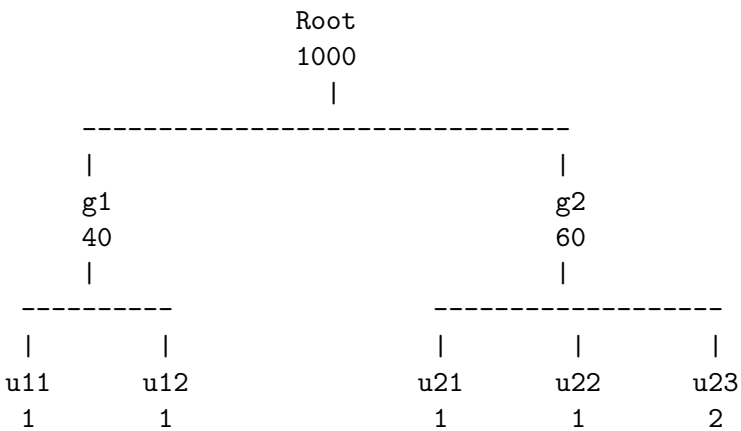
u11, u12	je 150,
u21, u22, u23	je 100.

Diese Arbeit nimmt uns nun der Share ab, indem er aus den den Usern und den Gruppen zugeteilten Shares eigene Shares (sogenannte Maschinen-Shares m_{share}) berechnet, die die Gruppenverteilung berücksichtigen. Er bestimmt dazu von der Spitze ("Root") ausgehend zunächst die Anteile der Gruppen, verteilt dann weiter deren Anteile auf die Untergruppen usw.. Dabei ist der Wert von Root ein reiner Skalierungsfaktor und repräsentiert 100

In Formeln:

$$m_{share}(node) = m_{share}(parent) \cdot \frac{share(node)}{share(node) + shares(geschwister)}$$

Beispiel:



$$
\begin{aligned}
m_{share}(g1) = &\quad 1000 * 40/(40+60) &&= 400 \\
m_{share}(g2) = &\quad \cdots &&= 600
\end{aligned}
$$

$$
\begin{aligned}
m_{share}(u11) &= 400 * 1/(1+1) &&= 200 = m_{share}(u12) \\
m_{share}(u21) &= 600 * 1/(1+3) &&= 150 = m_{share}(u22) \\
m_{share}(u23) &= 600 * 2/(2+2) &&= 300
\end{aligned}
$$

CONVEX-Implementierung des Share Schedulers
Der Fair Share Scheduler wurde von Convex dem normalen UNIX Scheduler "vorgeschaltet": Er setzt nur die Prioritäten der Prozesse, die Auswahl des aktiven Prozesses übernimmt weiterhin der UNIX-Scheduler.

Damit Systemprozesse immer den nötigen CPU-Anteil bekommen, wurden einige Besonderheiten eingeführt:

1. Prozesse von root werden nicht über den Share Scheduler mit CPU versorgt, sie bekommen soviel, wie sie brauchen.
2. Für die verschiedenen Dämonen wurden die Gruppen Dämon und Network eingeführt, die mit einem großen Share-Anteil versehen werden müssen.
3. Das Batchsystem tritt in Form einer Gruppe auf, die für jede queue einen User enthält. Batchjobs werden dann für diese User abgerechnet und schlagen nicht bei dem zu Buche, der sie abgesetzt hat. Damit kann man durch entsprechende Shares die CPU-Zeit zwischen Batch- und interaktivem Betrieb aufteilen.

Die für den Share Scheduler nötigen Daten stehen in dem File /etc/shares, das für jeden User u.a. den Share und die Usage enthält. Beim Einloggen eines Users wird eine Datenstruktur im Kernbereich angelegt (L-Node genannt), die diese Werte und andere enthält, die erst jetzt ermittelt werden können (s.u.). Nach dem Ausloggen muß der L-Node wieder gelöscht werden. Dies übernimmt ein Dämon, der Sharer-Dämon, der periodisch (im Abstand einiger Minuten) alle L-Nodes überprüft.

Nicht alle Funktionen des FSS sind bei Convex schon implementiert, es fehlt die Möglichkeit, auch andere Resourcen als die CPU-Zeit zu den Kosten zuzurechen.

User-Kommandos zum CONVEX Share Scheduler
Damit sich die User über die momentane Verteilung der Resourcen und die Ursachen dafür informieren können, werden einige Kommandos zur Verfügung gestellt, die die wichtigsten Kenngrößen auf verschiedene Weisen darstellen. Zu diesen gehören neben dem (intended) Share und der Usage:

- allocated Share
  Da in der Regel nicht alle User gleichzeitig eingelogged sind, wird die Maschine natürlich auch nur unter den aktiven Usern aufgeteilt. Der entsprechende Anteil heißt "allocated share".
  Formel:
  $$ashare(U) = \frac{share(U)}{\sum_{aktive(!)\,User\,U'} share(U')}$$
  Ein Problem ist hier die Bedeutung von "aktiver User": Alle eingelogten User zählen mit, auch wenn sie längere Zeit nichts eingeben.

- effective Share
  Der allocated Share berücksichtigt die Usage eines Users noch nicht; dies tut der effective Share.
  Formel:
  $$eshare(U) = \frac{\frac{share(U)}{nusage(U)}}{\sum_{aktiveUserU'} \frac{share(U)}{nusage(U)}}$$
  Dabei ist nusage = usage/share die normierte Usage eines Benutzers.

- charge

Die Usage zerfällt im Lauf der Zeit. Die charge dagegen sammelt alle Kosten auf, die ein User angesammelt hat, sie ist also ein Maß für den Gesamtverbrauch eines Users.

Die wichtigsten Kommandos im einzelnen:

- pl
  zeigt alle oben genannten Größen eines Users (default: für sich selbst)

- rates
  gibt intended und effective Share sowie den tatsächlichen momentanen (CPU-)Anteil für alle Gruppen oder für alle User in Form einer Balkengraphik aus

- charges
  zeigt die userunabhängigen Parameter von Share

- pwintf
  Ausgabe von /etc/shares-Einträgen, Syntax wie printf(C)

- sl
  wie su, ändert aber auch den L-Node, d.h. der Verbrauch wird der neuen User-Id zugerechnet.

Management des Share Schedulers
Die wichtigste Vorarbeit zur Inbetriebnahme des Share Schedulers ist das Erstellen einer Gruppenhierarchie (die nichts mit den UNIX-Gruppen zu tun haben muß) und die Vergabe der Share-Werte. Dann müssen alle Gruppen - incl. der Batchqueues, s.o., als neue User eingetragen werden. Das nu-Programm ist so angepaßt worden, daß es auch Share-Werte erfragt.

Auch für den System-Manager stehen neue Kommandos zur Verfügung, darunter:

- charge
  Änderung aller Share-Systemparameter
- lim
  Einträge in der Share-Database (für einzelnen Benutzer) ändern
- sharecf
  "front end" zu lim, ändert über ein Konfigurationsfile viele Einträge
- sharesh
  Ausgabe der Database als sharecf-File oder als Folge von lim-Kommandos
- remshent
  Entfernen eines Users aus der Database
- slxqt
  Absetzen eines Jobs unter dem Konto eines andern; wichtig für Prozesse, die unter Dämon oder Network laufen sollen.

Erste praktische Erfahrungen
Was bei der ersten Begegnung mit Share ins Auge fällt, ist das äußerst schmale Manual: 14 S. Concepts, 27 S. System Manager's Guide und 30 S. Programmer's Reference. Läßt dies schon nicht viel erwarten, so ist der zweite Eindruck noch schlechter: Neben "Äußerlichkeiten" wie Inkonsistenzen zwischen Synopsis und Text und fehlenden Seiten stört vor allem, daß weder die Funktionsweise des Share Schedulers noch zentrale Begriffe wie allocated/effective Share erklärt werden. Begriffe werden nicht einheitlich verwendet (share = intended/allocated/effective share) oder sogar mit zwei Bedeutungen (charges = Kosten für einen CPU-Tick oder = Gesamtsumme der Usage). Am verwirrendsten aber ist die Behandlung der Hierarchiestufen: In allen Beispielen ist der Share einer Gruppe gleich der Summe der Shares der User, eine Strategie, die der Zielsetzung des hierarchischen Share genau zuwiderläuft. Dies wird durch mißverständliche Formulierungen noch unterstützt, sodaß ich zweifelte, ob der Convex Share Scheduler überhaupt hierarchisch arbeitet.

Das Ausprobieren der verschiedenen Kommandos enthüllte einige Bugs, vor allem der pwintf-Befehl hat noch Probleme mit seiner komplexen Syntax, und dem nu fehlt noch eine wichtige (im Manual beschriebene) Abfrage. Problematisch ist auch die Darstellung diverser Werte als long long ints: 12-stellige Zahlen erhöhen nicht gerade die Übersichtlichkeit! Eine weitere Schwierigkeit ist die Einbeziehung der Systemprozesse mit hohen Shares: Sie liegen immer bei etwa 45%, wodurch alle interessanten Werte im Bereich weniger Prozent sind. Wünschenswert wäre eine Möglichkeit, solche Prozesse oder auch die Batchqueues auszuschließen und Prozentwerte auf die übrigen Prozesse zu beziehen. Außerdem sollte es möglich sein, einem laufenden Prozeß anzusehen, für welchen User er abgerechnet wird.

Grundsätzlich funktionierte aber alles, und so konnten die ersten Messungen mit 5 Testusern in 2 Gruppen (wie oben) mit verschiedenen Shares und Usages stattfinden. Die Ergebnisse zeigten, daß die Grundfunktionalität des Fair Share Schedulers gewährleistet ist; damit stellt der Convex Share Scheduler ein nützliches Instrument zur Verteilung der (CPU-)Resourcen dar. Allerdings sind in den Tools und in der Dokumentation noch so viele Macken, daß man Version 1.0 höchstens als Beta-Release bezeichnen kann. Hoffen wir also auf Verbesserungen in kommenden Versionen.

Quellen:
1. J. Kay und P. Lauder: “A Fair Share Scheduler”, Communications of the ACM, **31**(1988), 44 - 55.
2. Convex Manuals zum Convex Share Scheduler

Peter Junglas
Rechenzentrum der TU Hamburg-Harburg
Eißendorfer Str.38
2100 Hamburg 90

junglas@tu-harburg.dbp.de
junglas@tuhhco.rz.tu-harburg.de
junglas@tuhhco.uucp