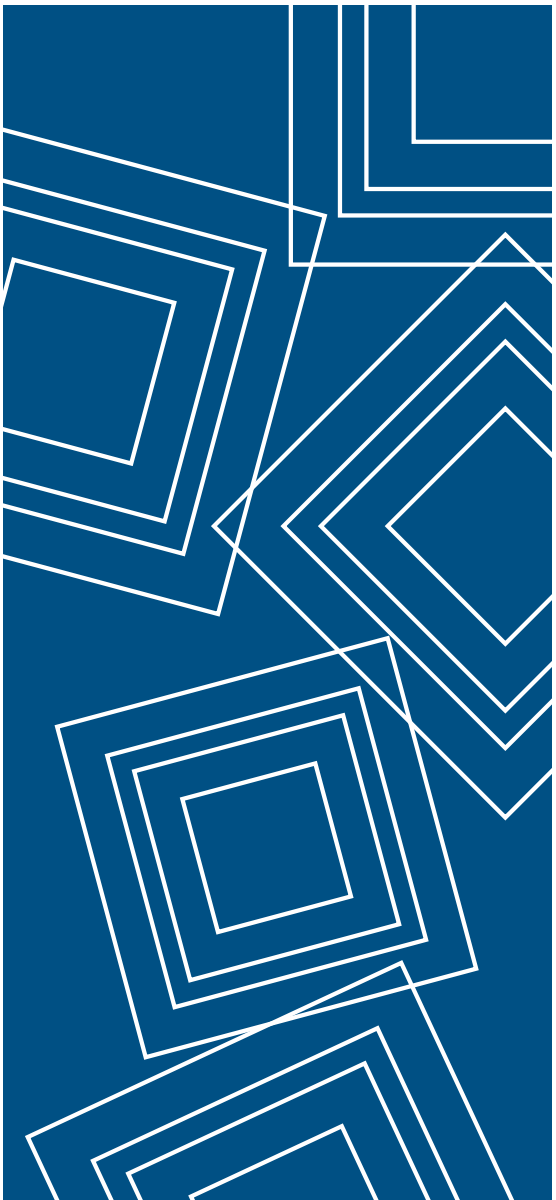
Eine Chance für Linux



Linux hat die Chance, das Tempo auf dem Markt skalierbarer Server vorzugeben. Als Schlüsselkomponente gilt dort eine effiziente Systemverwaltung. Zu deren Kernbestandteilen zählt ein modernes Performancemanagement, und das wiederum setzt eine verbesserte Instrumentierung des Betriebssystems voraus – die bislang fehlt. Kann sich Linux dieser Herausforderung stellen? ??????????????????????????

Dieser Beitrag entwickelt eine Vision, wie Linux in Zukunft aussehen könnte. Dafür kombinierter er verschiedene Aspekte der Performance-Instrumentierung, deren Zusammenspiel dem beiläufigen Beobachter vielleicht nicht offensichtlich ist. Als Ausgangspunkt für den Blick in die Zukunft soll eine kurze Rückschau auf das dienen, was heute vorhanden ist.

Eine kurze Geschichte der Instrumentierung

Der Begriff Instrumentierung wie ihn dieser Beitrag gebraucht, meint die Implementierung verschiedener Zähler im Kernel-Memory, die bestimmte Performance-Kennzahlen (Metriken) speichern – etwa I/O-Zähler, Zähler für ausgelagerte Speicherseiten (Page Swaps), den Auslastungsgrad der CPU und so weiter. Diese Zählerstände lassen sich auch in Raten umrechnen.

Das Konzept, die Interna eines Betriebssystem in dieser Weise zugänglich zu machen, ist nicht nur nicht neu für Linux, es ist sogar älter als Unix. Es geht auf das Multics-Projekt des MIT um 1965 zurück. Tatsächlich war eine der ersten Formen der Instrumentierung überhaupt das Pendant zum oft benutzten, aber selten verstandenen Load Average. (Der Load Avergae ist der gleitende Durchschnitt der Anzahl von Einträgen in der Run-Queue des Prozessors (1)).

Ursprünglich erfassten die Multics-Schöpfer solche Kennzahlen, um abzuschätzen, wie sich ihre Änderungen am Kernelcode und speziell am Scheduler auf die Performance auswirkten. Da Entwickler wie Dennis Ritchie (einer der Väter von Unix und der Programmiersprache C) am Multics-Kernel mitarbeiteten, war es nur natürlich, dass sie denselben Ansatz später auch beim Unix-Time-Share-Kernel wieder aufgriffen. Seitdem kamen nach und nach immer mehr Kennzahlen und Tools hinzu, die sie ad hoc ausgeben. Die Entwicklung von Linux verlief weitestgehend parallel.

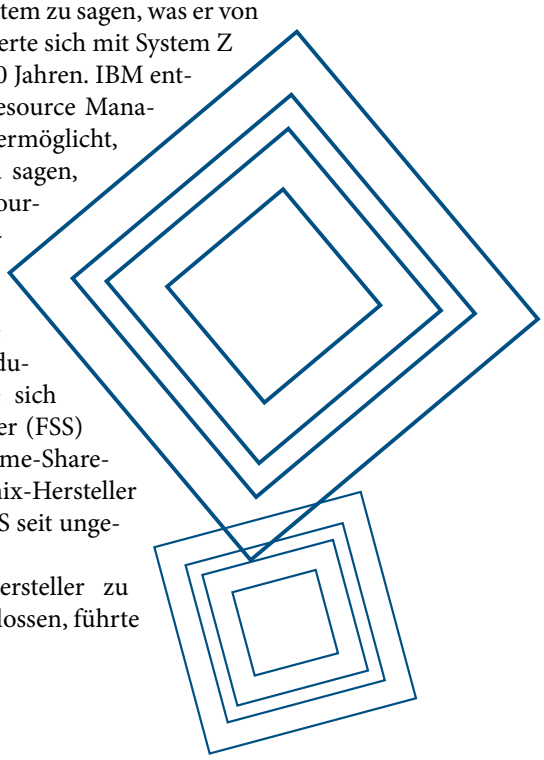
Ungeachtet der historischen Details ist der springende Punkt der, dass ursprünglich keine der Instrumentierungsmöglichkeiten für die Zwecke der Performance-Analyse oder Kapazitätsplanung gedacht war. Noch vor 20 Jahren konnte wahrscheinlich niemand vorhersehen, wie allgegenwärtig Unix- und Linux-Rechner heute etwa im Umfeld der webbasierten Applikationen sind. So hatte auch niemand eine Ahnung von den größeren Anforderungen an die Instrumentierung, die sich heute stellen.

Wer noch einen Schritt zurückgeht, stößt auf die Mainframes, die es bereits gab, lange bevor Unix oder Linux das Licht der Welt erblickten. Ja, die Mainframes: Im Gegensatz zu einer verbreiteten Meinung, sind sie keine Dinosaurier, die wegen der übermächtigen Konkurrenz der billigen Prozessoren inzwischen ausgestorben sind. Tatsächlich haben sie nicht nur diesen Ansturm überlebt (indem sie selber billiger wurden), sondern sie besitzen nach wie vor eines der besten Konzepte für die Betriebssystem-Instrumentierung und die ausgefeiltesten Management-Fähigkeiten, die heute auf dem Markt verfügbar sind.

Namentlich IBMs Betriebssystem System Z (vorher bekannt als Z/OS und davor als MVS) bietet durch seine Resource Measurement Facility (RMF) und die System Management Facility (SMF) eine einheitliche Darstellung von Daten für das Performance-Management und die Kapazitätsplanung. Vergleichbares gibt es weder in Unix noch in Linux. In dieser Beziehung lässt sich der Mainframe durchaus als Vorbild ansehen, dem es nachzueifern gilt.

Performance nach Plan

Bis vor gar nicht so langer Zeit ließen sich nur die Kennzahlen auswerten, die das Betriebssystem selbst gesammelt hatte – ganz gleich ob unter Unix, Linux oder Z/OS. Vereinfacht gesagt teilte das Betriebssystem dem Administrator über seine Ausgaben mit, wie es sich fühlte – aber es gab keine Möglichkeit für den Administrator, dem Betriebssystem zu sagen, was er von ihm erwartete. Das änderte sich mit System Z (oder MVS) vor rund 20 Jahren. IBM entwickelte den System Resource Manager, der es dem Admin ermöglicht, dem Betriebssystem zu sagen, welchen Anteil an Ressourcen jeder Nutzer beanspruchen darf. Diese neue Managementfunktion erforderte Änderungen am Scheduler und es entwickelte sich der Fair-Share-Scheduler (FSS) anstelle des alten Time-Share-Schedulers (TSS). Unix-Hersteller implementieren den FSS seit ungefähr zehn Jahren.



Während die Unix-Hersteller zu den Mainframes aufschlossen, führte

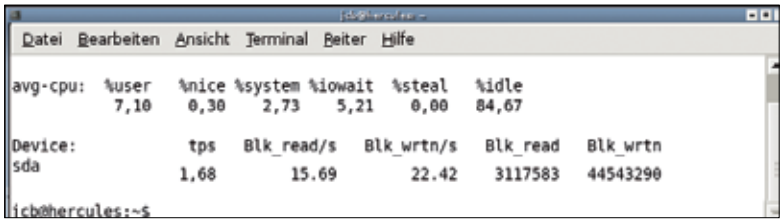


Abbildung 1: Das Kommando iostat gibt Disk-Statistiken aus, allerdings nur global und ohne Zeitstempel.

IBM ein noch ausgeklügelteres Konzept ein, den Goal-Mode-Scheduler (GSM). Er ermöglicht es, Performance-Ziele vorzugeben. Diese Ziele lassen sich entweder als Vorgaben für die Antwortzeit bestimmter Applikationen definieren oder als Zeitfenster, in dem bestimmte Batch-Workloads abgearbeitet sein sollen. Während das Betriebssystem die Aufgaben abarbeitet, meldet es nun zusätzlich, wie gut es die gestellten Ziele erreicht.

Performance-Instrumentierung unter Linux

Eine Momentaufnahme der Systemperformance unter Linux liefert standardmäßig das Kommando »procinfo«. Später fügte das »sysstat«-Paket eine Reihe traditioneller Unix-Performancetools hinzu wie »sar«, »mpstat« oder »iostat«.

Alle Performance-Kennzahlen, die diese Tools liefern, sind allerdings globale Kennzahlen (nicht per Prozess) und es fehlt ihnen – mit Ausnahme von »sar« – ein automatischer Zeitstempel.

Das Kommando »iostat« kombiniert Informationen über die CPU-Auslastung und I/O-Aktivitäten für Festplatten, »mpstat« liefert prozessorbezogene Statistiken. Sar sammelt eine breite Palette an Systemdaten, angefangen bei Daten zu einzelnen Prozessen, über das Paging und Swapping, Interrupts, Netzwerk-Aktivitäten, Speicher, CPU-Auslastung, TTY-Statistiken, I/O-Transferraten und anderes mehr. Sar unterscheidet sich dabei von den anderen Tools dadurch, dass es alle Daten

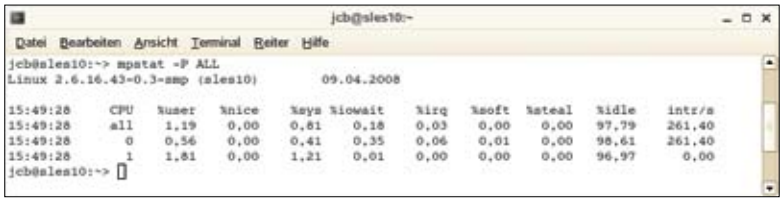


Abbildung 2: Das Mpstat-Kommando reportiert die Prozessor-Auslastung, hier auf einer Dual-Core-Maschine.

automatisch mit einem Zeitstempel versieht. Das Sysstat-Paket ist für Ein- oder Mehrprozessor-Systeme verfügbar.

Das Kommando »iostat« reportiert unter anderem die I/O-Transferrate pro Disk. Die Syntax ist: »iostat *interval count*«. Das Intervall-Argument gibt die Sample-Periode (also den Berichtszeitraum) vor und das Argument »count« die Anzahl der Reports, die »iostat« ausgibt, bevor es sich beendet. Voreingestellt ist »count = 1«. Die erste Ausgabe bezieht sich immer auf den Zeitraum seit dem Booten des Systems, alle nachfolgenden Reports gelten für die eingestellte Sample-Periode.

Die Disk »sda« im Beispiel der **Abbildung 1** erreichte einen Durschnittswert von 1,68 tps (I/Os) pro Sekunde. Dieser Wert berechnet sich so: Angenommen, die Sample-Periode sei T=30 sec und in dieser Zeit wurden 50 I/O-Operationen gemessen, dann ergäbe sich nach

$$\frac{C}{T} = \frac{50}{30} = 1.67 tps \quad (1)$$

was dem Wert in der dritten Zeile von **Abbildung 1** entspricht. Die Durchsatzrate ergibt sich also als Anzahl abgearbeiteter Anfragen »C« pro Zeiteinheit »T«.

Ähnliches gilt für die Ausgabe des Mpstat-Kommandos in **Abbildung 2** nach einem Aufruf von »mpstat -P ALL«. Es zeigt, dass der Rechner zwei mit »0« und »1« bezeichnete CPUs besitzt. Die durchschnittliche CPU-Auslastung lässt sich aus der zweiten Zeile ableiten: 100% - 97,79% = 2,21%. Das ist übrigens dasselbe wie %user + %nice + %system +%iowait + %irq = 1.19% + 0% + 0,81% + 0,18% + 0,03% = 2,21%. Die Berechnung führt der Kernel in gleicher Weise wie bei »iostat« aus. Beträgt das Sample-Intervall »T« beispielsweise 30 Sekunden und ist die CPU in dieser Zeit »B = 4,66« Sekunden belegt (das heißt nicht im Leerlauf), dann ergibt sich die CPU-Auslastung als derjenige Zeitanteil der Beobachtungsperiode, in der die CPU zu rechnen hatte.

$$\frac{B}{T} = \frac{4,66}{30} = 0,1553 \quad (2)$$

Das dieselbe CPU-Auslastung, wie sie die ersten Zeile von Abbildung 1 mit 100% - 84.67% = 15.33% berechnet. Abbildung 3 zeigt die Mpstat-Ausgabe eines 32-Wege-Multiprozessorsystems.

Das Problem mit diesen und ähnlichen Tools ist, dass das Ausgabeformat noch dasselbe ist wie vor 35 Jahren. Vom Standpunkt des Performance-Managements wäre aber zu wünschen, dass alle diese 100 bis 300 gesammelten Kennzahlen vom Benutzer auswählbar in einem leichter lesbaren Format präsentiert würden. Mehr noch, alle Metriken sollten einen automatischen Zeitstempel haben und am besten in einer Datenbank abgelegt sein, so dass ein Analyst sich leicht von einer Übersicht zu interessanten Details vorarbeiten könnte. Obwohl »sar« und neuere GUI-Tools manche dieser Anforderungen schon erfüllen, bleibt der Wunsch nach einem vereinheitlichten Ansatz unerfüllt, der auf allen Linux-Systemen in derselben Weise funktioniert.

Der erste Schritt in diese Richtung wäre die Definition einer allgemeinen Datenstruktur (eines Baums) für alle speicherresistenten Performancezähler. So etwas gibt es bereits in AIX und Solaris in Form der Datenstrukturen »rstat« beziehungsweise »kstat«. Eine noch weitergehende Lösung wurde bereits vor einem Jahrzehnt vorgeschlagen – der Beitrag kommt darauf zurück.

Eine Weggabelung?

Linux Torvalds wird mit der Aussage zitiert, er glaube, das die gegenwärtige Instrumentierung des Linux-Kernels auf reale Performanceprobleme ausreichend zugeschnitten sei. An anderer Stelle las der Autor Torvalds Aussage, er wolle Linux einfach und effektiv halten („lean and mean"). Im Licht der folgenden Gegenargumente erscheint dieses Design-Ziel jedoch als eine eher eingeschränkte Sicht auf das, was Linux erreichen könnte:

- Wie kann die Instrumentierung unter Linux ausreichend sein, wenn selbst die ältere Unix-Instrumentierung für ein modernes, mehrschichtiges Performance-Management nicht ausreicht?
- Die heute gebräuchliche Instrumentierung entstand nicht, um reale Performanceprobleme zu lösen (siehe oben). Sie ist sicher nötig, dadurch aber noch lange nicht hinreichend. Linux lebt nach wie vor mit dieser Altlast.
- Ich würde erwarten (naiverweise scheint's), dass ich inzwischen ein allgemeines Set von Performance-Metriken zur Hand habe, auf die ich über einheitliche Methoden auf allen Unix- oder Linux-Systemen zugreifen kann.

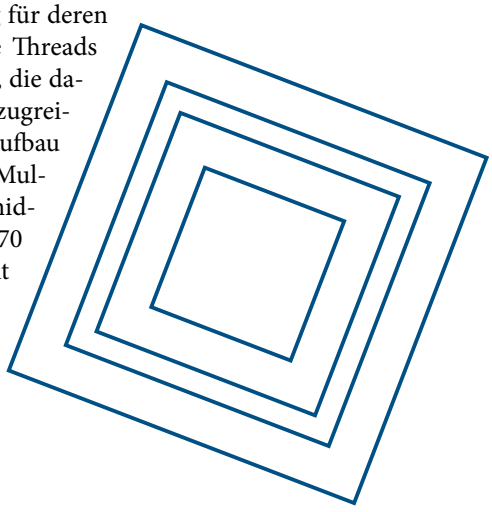
CPU	minf	mjf	xcal	intr	ithr	csw	icsw	migr	smtx	srw	syscl	usr	sys	wt	idl
0	2	0	4191	7150	6955	1392	93	374	573	14	1433	78	22	0	0
1	2	0	179	11081	10956	1180	132	302	1092	13	1043	79	21	0	0
2	1	0	159	9524	9388	1085	141	261	1249	14	897	79	21	0	0
3	0	0	3710	10540	10466	621	231	116	1753	2	215	70	29	0	0
4	5	0	28	355	1	2485	284	456	447	30	2263	77	23	0	0
5	5	0	25	350	1	2541	280	534	445	26	2315	78	22	0	0
6	3	0	26	331	0	2501	267	545	450	28	2319	78	22	0	0
7	2	0	30	292	1	2390	232	534	475	23	2244	77	22	0	0
8	4	0	22	265	1	2188	220	499	429	26	2118	75	25	0	0
9	2	0	28	319	1	2348	258	513	440	26	2161	76	24	0	0
10	4	0	23	308	0	2384	259	514	430	22	2220	76	24	0	0
11	4	0	27	292	0	2366	237	518	438	30	2209	77	23	0	0
12	11	0	31	314	0	2446	253	530	458	27	2290	78	22	0	0
13	4	0	31	273	1	2334	223	523	428	25	2261	79	21	0	0
14	12	0	29	298	1	2405	247	521	435	25	2286	78	22	0	0
15	4	0	32	330	1	2445	272	526	450	24	2248	77	22	0	0
16	5	0	28	271	0	2311	219	528	406	29	2188	76	23	0	0
17	4	0	23	309	1	2387	253	537	442	25	2234	78	22	0	0
18	3	0	25	312	1	2412	257	534	449	26	2216	78	22	0	0
19	3	0	29	321	1	2479	262	545	462	31	2287	78	22	0	0
20	14	0	29	347	0	2474	289	541	457	24	2253	78	22	0	0
21	4	0	29	315	1	2406	259	534	469	24	2240	77	22	0	0
22	4	0	27	290	1	2406	243	531	480	25	2258	77	22	0	0
23	4	0	27	286	1	2344	235	531	445	26	2240	77	22	0	0
24	3	0	30	279	0	2292	228	518	442	22	2160	77	23	0	0
25	3	0	26	275	1	2340	227	538	448	25	2224	76	23	0	0
26	4	0	22	294	1	2349	247	529	479	26	2197	77	23	0	0
27	4	0	27	324	1	2459	270	544	476	25	2256	77	23	0	0
28	4	0	25	300	1	2426	249	549	461	27	2253	77	23	0	0
29	5	0	27	323	1	2463	269	541	447	23	2277	77	22	0	0
30	2	0	27	289	1	2386	239	535	463	26	2222	77	23	0	0
31	3	0	29	363	1	2528	304	525	446	26	2251	76	23	0	0

Abbildung 3: Das Kommando »mpstat« auf einem 32-Wege-Multiprozes-

- Es gab bereits verschiedene Versuche, die Performance-Instrumentierung sowohl für Unix wie für Linux zu verbessern und zu standardisieren.

Der Beitrag kommt auf alle diese Punkte zurück. Weiter: Desktops, Laptops und mobile Geräte sind heute durch die Multicore-Technologie immer öfter Multiprozessor-Rechner, sie verwenden also mehrere CPUs auf einem Chip. Man kann annehmen, dass diese Rechner – mit Ausnahme der Server – weiter vorwiegend Single-Thread-Applikation ausführen. Mit anderen Worten: Jeder Core ist einer anderen Applikation zugeordnet. Für diesen Fall würde ich zustimmen, dass Linux einfach und effektiv bleiben soll.

Betrachtet man jedoch die Mid-Range- und High-End-Server, ergibt sich ein ganz anderes Bild. Eine Schlüsselanforderung für deren Skalierbarkeit ist, dass sie viele Threads parallel abzuarbeiten vermögen, die dabei auf gemeinsamen Speicher zugreifen. Meine Erfahrung beim Aufbau kommerzieller symmetrischer Multiprozessorsysteme bei Pyramid-Siemens hat mir gezeigt, dass 70 Prozent der Entwicklungszeit nötig waren, um eine gute Skalierbarkeit der Applikationen (zum Beispiel Datenbanken)



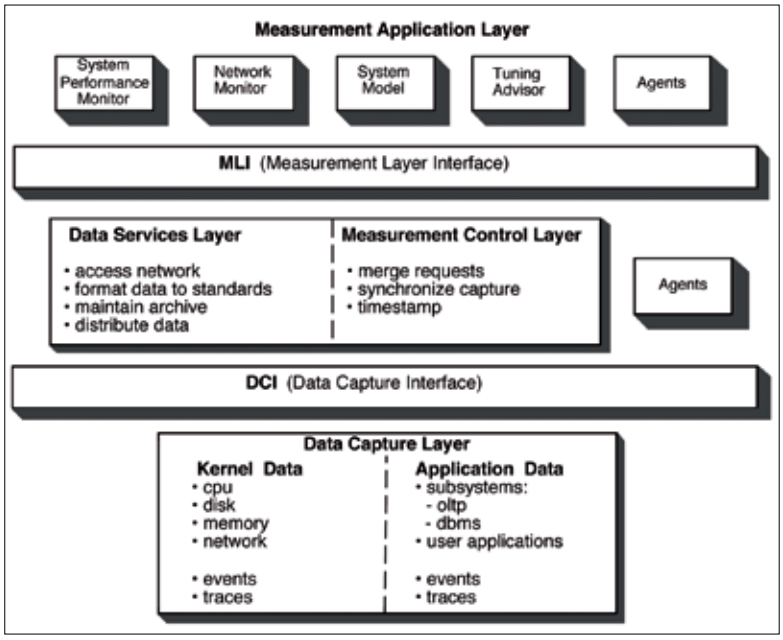


Abbildung 4: Das UMA-Referenzmodell definierte bereits in den 90er Jahren ein einheitliches Schema für den Zugriff auf Performance-Daten.

zu erreichen. Das heißt: Ein effizienter symmetrischer Kernel benötigt mehr Code (und nicht etwa weniger), um vitale Kontrollstrukturen zu implementieren. Das betrifft beispielsweise

- Prozessor-Affinität und -Auslastung,
- Mutex-Locking,
- Spin-Waiting.
- Preemption Control

Diese Kontrollstrukturen ermöglichen einen effizienteren, seriellen Zugriff auf gesharte Ressourcen und erlauben dadurch eine dramatische Verbesserung der Skalierbarkeit von SMP-Applikationen. Dasselbe gilt für Linux-basierte SMPs und der neue Scheduler im 2.6er Kernel unterstützt deshalb auch bereits Prozessor-Affinität und Preemption Control. Mehr noch, solche Steuerungsmöglichkeiten erhalten bei der Entwicklung von Applikationen für Multicore-CPUs eine noch höhere Bedeutung, denn diese Prozessoren sind immer die billigste, aber nicht die skalierbarste Variante. Die Skalierbarkeit der Software und des Betriebssystems sind dann der einzige Weg, die Beschränkungen der Hardware aufzuwiegen.

Betriebssystem-Hersteller vermeiden die Investitionen in alternative Möglichkeiten der Instrumentierung, solange sie keinen rentablen Bedarf dafür sehen. Gleichzeitig melden System-Administratoren keinen Bedarf an einer solchen Instrumentierung an, solange sie davon keine Vorstellung haben.

Der Linux-Kernel enthält ab Version 2.6.23 einen modularisierten Scheduler mit einer neuen Komponente namens Completely Fair Scheduler (CFS). Es handelt sich um eine komplette Neuentwicklung des Linux Task-Schedulers, die sowohl auf Desktop- wie auch auf Server-Workloads abzielt. Wie schon erwähnt, finden sich Einzelbenutzer-Desktops und Mehrbenutzer-Server aber an entgegengesetzten Enden des Skalierbarkeits-Spektrums.

Eine andere Dimension bilden interaktive und Batch-Workloads. Erstere erfordern optimale Antwortzeiten, letztere einen optimalen Durchsatz. Historisch gesehen zielten die Mainframe-Betriebssysteme vor allem auf das Batch-Prozessing großer Datenmengen ab, die interaktive Verarbeitung kam erst später dazu. Die Entwicklung von Unix nahm genau den entgegengesetzten Weg, was auch auf die Ausrichtung der jeweiligen Scheduler abgefärbt hat. Der neue CF-Scheduler versucht nun – so erläutert es die Dokumentation (3) – die CPU immer der Task mit dem höchsten Bedarf an Prozessorzeit („gravest need“) zuzuteilen und das bewirkt, dass jeder Prozess seinen fairen Anteil erhält. Das ursprüngliche Ziel des Vorläufers, des Rotating Staircase Deadline Scheduler (RSDL) war es, die Antwortzeiten von Desktop-Applikationen zu verbessern, aber auch Batch-Tasks mögen davon profitieren.

Wie sich zeigt, ergibt sich leicht ein Konflikt in der Frage, auf welche Workloads und welche Marktsegmente sich Linux nun ausrichten soll. Vielleicht ist es Zeit für die Linux-Community den berühmten, unfreiwillig komischen Spruch des Basketballstars Yogi Berra ernst zu nehmen: „Wenn du an eine Weggabelung kommst, dann nimm sie.“ Die Frage, ob man die Gabelung als Aufteilung in Server- und Desktop-Releases interpretieren sollte, oder als Trennung mittels unterschiedlicher Kernel-Module liegt außerhalb der Reichweite dieses Artikels.

Die Universal Measurement Architecture

Es ist ein gut gehütetes Geheimnis, dass eine Gruppe von Unix-Herstellern bereits zwischen 1990 und 1995 ein einheitliches Framework für die Erfassung und Verteilung von Performance-Daten entwickelt hatte: die Universal Measurement Architecture, kurz UMA. Zu den Entwicklern gehörten damals Firmen wie Amdahl (heute Fujitsu), BGS (heute Teil von BMC), Hitachi, HP, IBM, NCR, OSF, Sequent (gehört heute IBM) und einige andere. Die breite Zusammenarbeit auf organisatorischer Ebene war nötig, damit die Architektur die Schwierigkeiten beim Sammeln der Daten in einer verteilten Umgebung, auf verschiedenen Ebenen und über

viele Applikationen hinweg überwinden konnte. Das Ergebnis war damit sogar effektiver als das bekannte Simple Network Management Protocol SNMP.

Ironischerweise kam die Motivation für UMA aus der Mainframe-Welt. Amdahl stellte zu dieser Zeit Mainframes her, die IBMs Betriebssystem MVS benutzten. Gleichzeitig wollte man aber auch Rechner im Telco-Segment verkaufen, wo die Vorgabe allerdings Unix hieß. Deshalb entwickelte Amdahl eine eigene Unix-Version namens Unix Time Share, kurz UTS. Da Amdahl gleichzeitig auch die SMF-Instrumentierung unter MVS kannte (siehe oben), entwarf man UMA, um diese Möglichkeiten im eigenen Unix UTS nachzurüsten. Die später generalisierte UMA-Spezifikation findet sich heute bei der Open Group (4).

Das UMA-Referenz-Modell definiert vier Layer und zwei Interfaces (Abbildung 4):

- Der unterste Data Capture Layer ist für die Sammlung der Rohdaten verantwortlich. Seine Architektur ermöglicht zusammen mit dem Data Capture Interface (DCI), dass eine einzelne Sammelstelle oberhalb des DCI die Daten aus verschiedenen Quellen in Empfang nehmen kann, was die Synchronisierung vereinfacht.
- Das Data Capture Interface findet sich zwischen dem Data Capture- und dem Measurement Control Layer. Es stellt die Mittel bereit, um die gesammelten Daten weiterzugeben, beispielsweise an Datenbanken, ohne dass sich bereits bestehende Applikationen darum kümmern müssten.
- Der Measurement Control Layer managet die Zeitsteuerung und Synchronisation der Datensammlung.
- Der Data Service Layer nimmt über sein Measurement Layer Interface (MLI) Anforderungen einer Mess-Applikation (Measurement Application Program, MAP) entgegen (Abbildung 5) und gibt die Daten an das von der MAP bestimmte Ziel weiter – das mag die MAP selber sein, eine Datei oder der UMA Data Storage (UMADS)

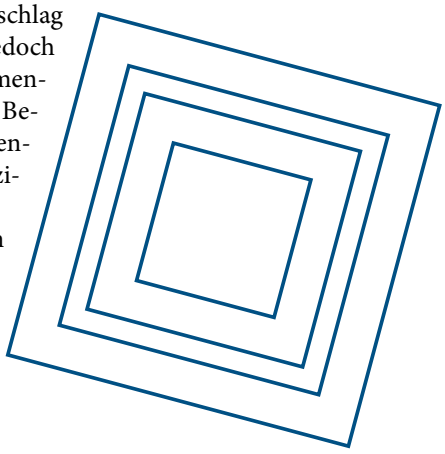
Zugriff via Speicher

Das »/dev/kmem«-Interface war geschichtlich betrachtet das erste Interface, über das Unix-Programme Performancedaten vom Kernel bezogen. Kennt ein Programm den Namen einer bestimmten Datenstruktur, kann es deren virtuelle Adresse in der Symboltabelle des Bootable Object File nachschlagen, und dann via »/dev/kmem« ihre Werte aus dem zugehörigen Speicherbereich auslesen. Der Vorteil dieses Ansatzes ist seine Allgemeingültigkeit: Lässt sich die Adresse ermitteln, ist es immer möglich, auch auf die Werte zuzugreifen. Aber diese Allgemeingültigkeit ist zugleich ein Nachteil: Da sich so gut wie jede Datenstruktur für Performancedaten eignet, benutzen die Entwickler auch tendenziell jede Struktur ohne Rücksicht darauf, ob sie sonst irgendwer irgendwie unterstützt. Und das wiederum macht es schwierig, Performance-Applikationen lauffähig zu halten, sobald sich die Datenstrukturen ändern.

Abbildung 5: Ein Measurement Application Program, das die Prozessorauslastung als Zeitreihe und Histogramm zeigt. Ein Schieberegler ermöglicht es, historische Daten einzublenden.

In jüngerer Zeit haben besondere Strukturen für Performancedaten wie »kstat« für Solaris oder »rstat« für AIX die Datensammlung portabler gemacht. Einen ähnlichen Vorschlag gab es auch für Linux (5), jedoch wurde er bislang nicht implementiert. Um diese Probleme und Beschränkungen zu umgehen enthielt die UMA-Definition Spezifikationen wie:

- eine Data Pool Specification für gebräuchliche Metriken,
- das Data Capture Interface (DCI) für Datenquellen,



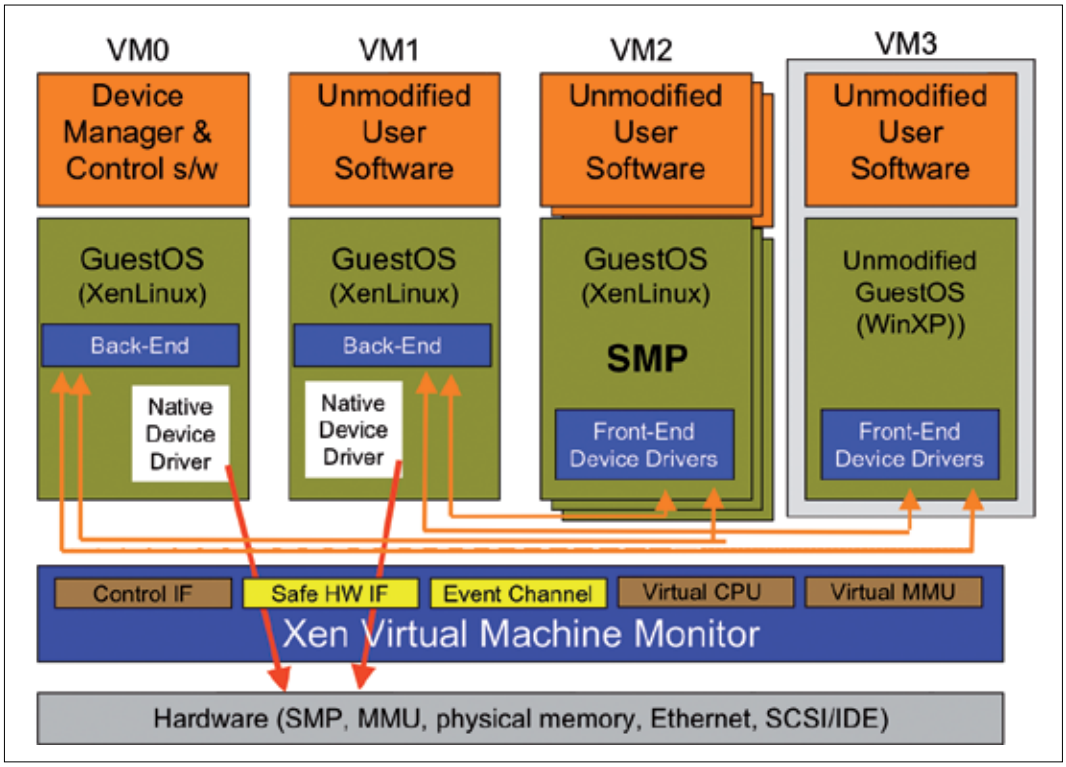


Abbildung 6: Die Architektur von Xen im Überblick. Der Hypervisor bildet die Schnittstelle zwischen Gastbetriebssystemen und Hardware.

- das Measurement Layer Interface (MLI) für Performance-Tools.

Chance verpasst

Nach der Schilderung der Vorzüge, die der UMA-Ansatz offensichtlich hatte, stellt sich zwangsläufig die Frage: Warum bieten die Unix-Hersteller, die UMA entwickelt haben, es immer noch nicht an?

Ein Teil der komplizierten Antwort ist, das UNIX-Hersteller kommerzielle Unternehmen sind, die nur widerwillig neue Instrumentierungsmöglichkeiten unterstützen, solange sie keine Nachfrage dafür sehen, die ihnen einen schnellen Rückfluss ihrer Entwicklungskosten verspricht. Außerdem versäumte es die UMA-Arbeitsgruppe, ihre Entwicklungsergebnisse zu vermarkten. Stattdessen nahm sie einfach an, die UMA-Spezifikation würde sich schon verkaufen, wenn sie erst fertiggestelllt wäre.

Auf der anderen Seite tappten die Systemadministratoren im Dunkeln und konnten keine bessere Instrumentierung einfordern, weil sie von den Entwicklungsbemühungen nichts wussten. UMA war eine Hersteller-Konzeption, keine Anwender-Konzeption und endete als Lösung, die nach dem Problem sucht. Das Problem jedoch ist immer noch da – und UMA als möglich Lösung gleichzeitig weithin unbekannt.

Ob man nun mit den technischen Details der UMA-Spezifikation übereinstimmt oder nicht, Fakt ist, dass das Konzept eine Möglichkeit beschreibt, wie ein vereinheitlichtes Interface für die Instrumentierung sowohl unter Unix wie unter Linux zu realisieren wäre, sodass der Administrator sich nicht mehr darum kümmern bräuchte, ob seine Performance-Skripte auf beiden Plattformen unverändert laufen. In der Rückschau ist es eher deprimierend, wenn man sich vor Augen hält, dass viele Leute dachten, wir würden um die Jahrtausendwende mit unseren Rechnern in natürlicher Sprache reden, während wir tatsächlich noch nicht einmal garantieren können, dass ein einfaches Skript auf zwei Linux-Derivaten läuft.

Jede Virtualisierung dreht sich um eine Illusion. Sie dem ahnungslos glücklichen User zu verkaufen, erscheint vernünftig. Aber es sollte verboten sein, dieselbe Illusion unter Systemadministratoren zu verbreiten.

Virtual Machine Managers und Hypervisors

Die Virtualisierung (von Diensten) ist ein heißes Thema, bietet sie doch zahlreiche Möglichkeiten für die Server-Konsolidierung, verteilte Web-Services, die gemeinsame Installation verschie-

dener Dienste auf einem Host, für die Isolation von Diensten, die Mobilität von Applikationen oder die Sicherheit – und alles, ohne dass sich der Admin zu viele Gedanken darüber machen muss, wie das zu erreichen wäre. Jedenfalls ist das der Slogan des Marketing und zum großen Teil auch die allgemeine Auffassung. Aber wie jeder weiß, der schon einmal versucht hat, die Performance eines Virtual Machine Managers zu tunen: Die Wirklichkeit kann ein wenig anders aussehen.

Für alle, die mit der Materie nicht vertraut sind: Virtualisierung, in dem Sinn, wie der Begriff hier benutzt wird, bedeutet, dass eine beliebige Anzahl unterschiedlicher Betriebssysteme (beispielsweise Windows XP, MacOS X und Linux) parallel auf derselben Plattform unter der Aufsicht eines allgewaltigen Virtual Machine Monitors oder Hypervisors laufen können. Er stellt das Interface zwischen den Betriebssystemen und der darunterliegenden Hardware zur Verfügung. Während jeder Benutzer die Betriebssystem-Instanzen sieht, bleiben die Details des Hypervisors normalerweise verborgen. Wie alles Virtuelle, handelt auch der Virtual Machine Manager mit einer Illusion. Die allerdings entpuppt sich schnell als Quelle realer Probleme, weil dem Administrator zu viele wichtige Einzelheiten verborgen bleiben – denn eine angemessene Instrumentierung fehlt.

Wie müsste sie aussehen? Dazu ist es nötig einen Blick auf die grundlegenden Funktionsprinzipien eines Virtual Machine Managers zu werfen.

Time Share und mehr

Die Partitionierung der Ressourcen einer physischen Maschine, die dadurch gleichzeitig verschiedene Betriebssystem-Instanzen unterstützen soll (**Abbildung 6**), birgt eine Reihe von Herausforderungen. Erstens müssen die Betriebssysteme strikt voneinander isoliert sein. Es wäre nicht zu akzeptieren, dass das eine ein anderes nachteilig beeinflusst. Das gilt erst recht, wenn die Instanzen Usern gehören, die sich gegenseitig nicht völlig vertrauen. Zweitens ist es notwendig, eine Vielzahl verschiedener Gastsysteme zu unterstützen, um der Vielfalt populärer Applikationen zu entsprechen. Drittes – und am wichtigsten unter Management-Gesichtspunkten – sollte der von jedem Gast-System erzeugte Performance-Overhead natürlich möglichst klein sein.

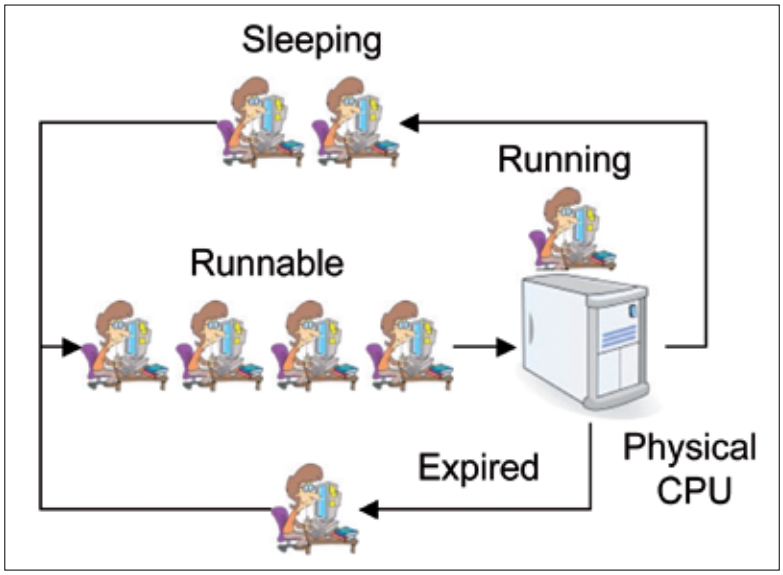


Abbildung 7: Das Funktionsprinzip des TS-Schedulers: Die Prozesse der Run-Queue erhalten nacheinander eine Zeitscheibe zugeteilt. Bleibt danach noch etwas zu tun, stellen sie sich wieder hinten an. Wer im Moment nicht arbeitsbereit ist, legt sich schlafen.

Historisch betrachtet waren alle Linux- und Unix-Scheduler ursprünglich vom Typ Time-Share-Scheduler (TSS). Wie oben erläutert implementieren die aktuellsten Versionen des 2.6er Kernels nun andere Scheduler-Klassen wie den CFS (in »kernel/sched_fair.c«) oder einen Real-time-Scheduler (in »kernel/sched_rt.c«). Ein weiterer Scheduler-Typ, den dieser Artikel gleich am Anfang erläutert hat, ist allerdings bis jetzt noch nicht in Linux verwirklicht: der Fair-Share-Scheduler (FSS). Der FSS erfordert die explizite Ressourcenvergabe an User durch den Administrator. Mehr noch: Der FSS – häufig findet das wenig Beachtung – bildet auch den Unterbau für jeden Virtual Machine Manager (mehr dazu bei (6)). Um den FSS besser zu verstehen, ist es hilfreich sich zuvor noch einmal vor Augen zu führen, wie sein Vorgänger TSS arbeitet.

Zweck des TSS ist es einfach, jedem Benutzer die Illusion zu geben, er sei die einzige Person, die auf dieser physischen Plattform arbeitet. In Linux befindet sich jeder User-Prozess in einem von drei möglichen Zuständen: Running, Runable oder Sleeping. Läuft ein Prozess (Status Running), befindet er sich im unteren Teil der **Abbildung 7** in der physischen CPU. Ist der Prozess zwar ausführbar (Status Runable), läuft aber nicht, dann findet er sich in der Prozessor-Warteschlange oder Run-Queue, in der Abbildung unmittelbar links von der CPU. Wurde ein

Mainframes sind keine Dinosaurier, die im Konkurrenzkampf mit dem Mikroprozessor ausstarben. Sie leben heute nach wie vor als sehr leistungsfähige Datenverarbeitungsmaschinen. Linux hat die Chance aufzuschließen, vielleicht sogar zu überholen.

Prozess nicht komplett abgearbeitet, wenn sein Zeitquantum abgelaufen ist (beispielsweise 10 ms oder 50 ms bei VMware), dann reiht er sich wieder an das Ende der Warteschlange ein. Ist der Prozess jedoch momentan gerade nicht ausführungsbereit, etwa weil er auf das Ende einer I/O-Operation wartet, dann legt er sich vorerst schlafen.

Im Unterschied dazu vermittelt der FSS, den schematisch die **Abbildung 8** zeigt, jedem Anwender der beiden Gruppen A und B die Illusion, er habe eine komplette Plattform für sich (eine virtuelle Maschine), deren Performance – also beispielsweise die CPU-Geschwindigkeit – sich nach den ihm zugeteilten Ressourcen richtet. Die Anwartschaften vergibt der Administrator durch Zuweisung von Shares, ganz ähnlich der Ausgabe von Aktien, die Dividendenansprüche begründen. Die physische Servicezeit »Sguest« für jeden Gastprozess (der isoliert von den anderen läuft) wird zu einer virtuelles Servicezeit:

$$S^{v}_{guest} = \frac{S_{guest}}{\varepsilon_{guest}}$$ (2)

Listing 1: Ressource-Zuweisung im Pseudo-

VM Share Scheduling: Pollt alle 4000 ms, um die Verwendung der physischen CPU mit den Zuweisungen an die User zu vergleichen (Abbildung 5).

```
for(i = 0; i < USERS; i++) {
usage[i] *= decayUsage;
usage[i] += cost[i];
cost[i] = 0;
}
```

VM Priority Adjustment: Pollt alle 1000 ms und braucht den internen FSS Prozess-Priority Wert auf..

```
priDecay = Value in [0..1];
for(k = 0; k < PROCS; k++) {
sharepri[k] *= priDecay;
}
priDecay = a * p_nice[k] + b;
```

Time Share Scheduling: Pollt mit jedem physischen Prozessor-Tick, um die Prozess-Priorities anzugleichen (Abbildung 4).

```
for(i=0; i<USERS; i++) {
sharepri[i] += usage[i] * p_active[i];
}
```

Die virtuelle Zeit läuft langsamer oder schneller als »Sguest«, je nachdem welche Ressourcen der Gast erhalten hat.

Der Xen-Server (**Abbildung 3**) verwendet standardmäßig eine Spielart des FSS die Borrowed Virtual Time (BVT) heißt. Es gibt jedoch auch andere Optionen, zum Beispiel das Real-Time-Scheduling. Der BVT-Scheduler bietet FSS-Scheduling für den Prozessor auf der Grundlage von Gewichten. Jeder ausführbare Gast erhält dabei entsprechend seinem Gewicht einen Anteil an den CPU-Ressourcen. Zum Beispiel könnte eine einzelne VMware-Instanz per Default 1000 Shares erhalten **(6)**. Diese Zuweisungen können die Gesamtperformance signifikant beeinflussen.

Der FSS führte eine Scheduling-Superstruktur ein, die auf dem konventionellen TS-Scheduler aufsetzt und Prozesse und User mit Ressource-Zuweisungen so verbindet, wie es der stark vereinfachte Pseudocode in **Listing 1** illustriert.

Das Polling auf Prozessebene ist im Wesentliche dieselbe Vorgehensweise wie beim TS-Scheduler, wogegen das VM-Share-Polling den Ressourcenverbrauch per Prozess kontrolliert.

Ressourcen kappen

Für jedes Paar von Benutzern mit den Zuweisungen E_1 und E_2 lässt sich das Ziel des FSS-Schedulers beschreiben als

$$\lim_{t\to\infty} \frac{\rho_1}{\rho_2} = \frac{\varepsilon_1}{\varepsilon_2}$$ (2)

Dabei stehen E_1 und E_2 für die jeweilige Ressourcennutzung durch die Anwender. Mit anderen Worten: Das Langzeitziel des FSS besteht darin zu versuchen, die prozentuale Ressourcen-Inanspruchnahme durch den Benutzer mit den ihm zugewiesenen Ressourcen-Anteilen in Einklang zu bringen. Warum möchte man das tun? Zwei der Gründe sind das Ressourcen-Management und die Kostenrechnung. Letztere ist Linux-Anwendern nicht nur unbekannt, sondern oft ein Dorn im Auge. Die Wirklichkeit sind allerdings: CPU-Cyclen sind niemals umsonst und manche Rechenzentren machen ihre Manager sogar direkt für den Ressourcenverbrauch verantwortlich. Es gibt allen Grund anzunehmen, dass sich dieser Trend weiter fortsetzt, gerade im Bemühen um eine energieffizientere und damit grünere IT.

Angenommen, einem Benutzer wurden 10 Prozent der Prozessor-Ressourcen zugeteilt, indem er 10 von 100 systemweiten Prozessor-Shares erhält. Dieser Benutzer sei momentan der einzige im System. Sollte er unter diesen Umständen die CPU zu 100 Prozent beanspruchen dürfen? Die meisten Administratoren glauben, das es sinnvoll sei, ihm die CPU ganz zu überlassen – weil sonst der Server zu 90 Prozent leer liefe.

Aber kann ein Benutzer überhaupt die CPU zu 100 Prozent belegen, wenn er nur für 10 Prozent Shares besitzt? Er kann, wenn der FS-Scheduler nur aktive Shares in seine Rechnung einbezieht. In diesem Fall würde der einzige Benutzer zehn von zehn aktiven Shares besitzen und dürfte deshalb die CPU ganz für sich beanspruchen.

Die natürliche Neigung, ansonsten ungenutzte Ressourcen in Beschlag zu legen, gründet sich auf zwei Annahmen:

1. Der Benutzer zahlt nicht für die Prozessor-Ressourcen. Hat sein Manager dagegen nur so viel Budget, um 10 Prozent der Prozessor-Kapazitäten zu bezahlen, dann wäre es finanziell nicht wünschenswert, dieses Limit zu überschreiten.
2. Den Benutzer kümmern die Service-Ziele nicht. Es ist ein Naturgesetz, dass Benutzer sich über Antwortzeiten beklagen. Gibt es nun Perioden, zu denen die Antwortzeiten deutlich kürzer ausfallen, dann definieren sie automatisch das zukünftige Service-Ziel. Dadurch bergen die vom Benutzer wahrgenommenen Änderungen der Performance ein Problempotential.

Unter Umständen kann es daher sinnvoller sein, die Ressourcen zu kappen. Das Beispiel eines Geschäftsempfangs soll das verdeutlichen. Angenommen zu einer Firmenfeier in einem Hotel wurden 200 Personen eingeladen. Der Veranstalter bezahlt 2000 Dollar pro Raum mit Catering. Jeder Raum fasst bequem 100 Personen.

- Kappung aktiviert: Unerwartet kommen 220 Gäste und es ist ein bisschen eng (**Abbildung 10**). Doch der Veranstalter bekommt genau das, wofür er gezahlt hat und es wäre unklug von ihm, wegen der 20 zusätzlichen Gäste einen ganzen Raum zusätzlich zu mieten. Diese Situation entspricht einem FSS-Scheduler mit aktivierter Kappung (capping).
- Kappung deaktiviert: Das Hotel erlaubt dem Veranstalter einen dritten Raum solange mitzubenutzen, wie eine andere Gesellschaft, die diesen Raum gemietet hat, noch nicht eingetroffen ist. Das lindert vorübergehend die Enge, doch die Gäste fühlen sich möglicherweise erst recht unwohl, wenn sie später den dritten Raum wieder räumen und mit den ursprünglich angemieteten beiden Räumen auskommen müssen (**Abbildung 11**). Diese Situation entspricht dem FS-Scheduler mit abgeschalteter oder nicht implementierter Ressourcen-Kappung.
- Kappung unbenutzt: Eine dritte Möglichkeit wäre: Nur 100 Gäste kommen (**Abbildung 12**). Der Veranstalter hat mehr bezahlt als er braucht, aber die Gäste fühlen sich wohl. Diese Situation entspräche dem TS-Scheduler, die Kappungsgrenze wird nicht erreicht.

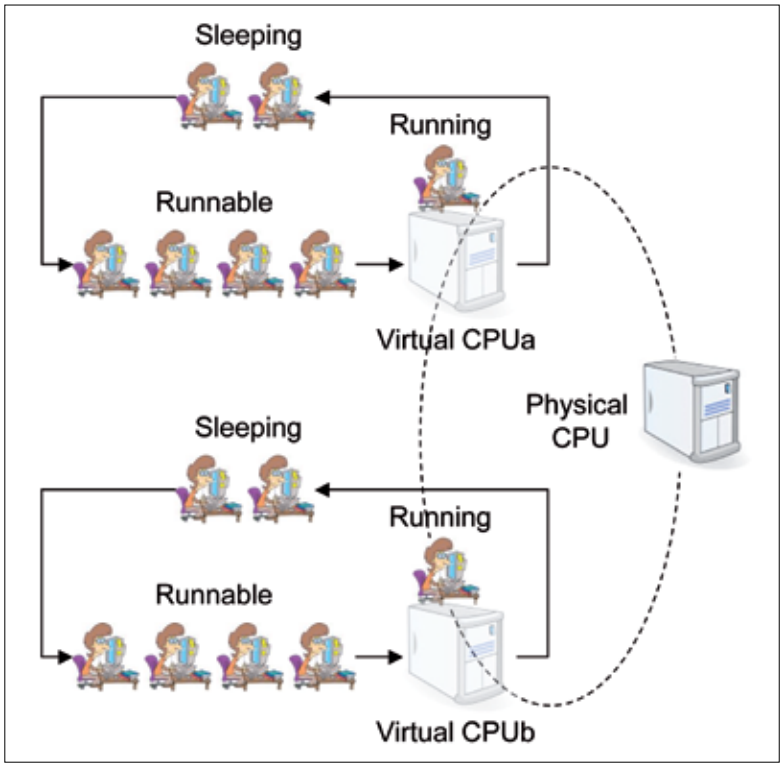


Abbildung 8: Das Funktionsprinzip des FSS. Jeder Anwender hat eine eigene Run-Queue, über die Zurteilung der CPU entscheiden Anwartschaften, die der Admin den Usern zuteilt.

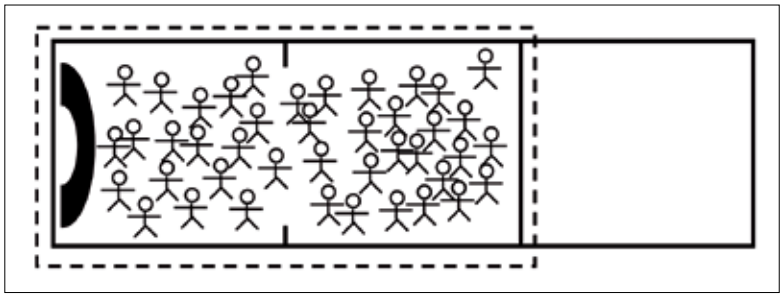


Abbildung 9: Mehr Gäste als erwartet drängen sich in zwei Räumen. Die Platzressourcen sind gekappt.

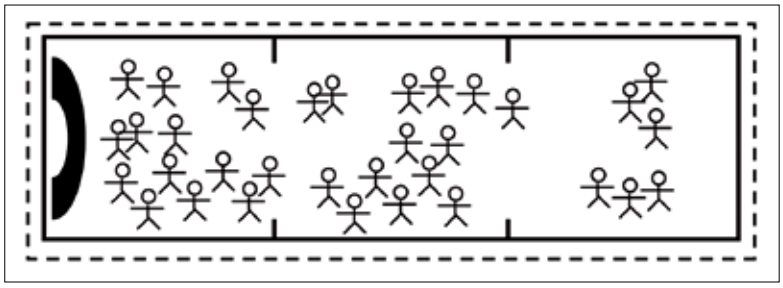
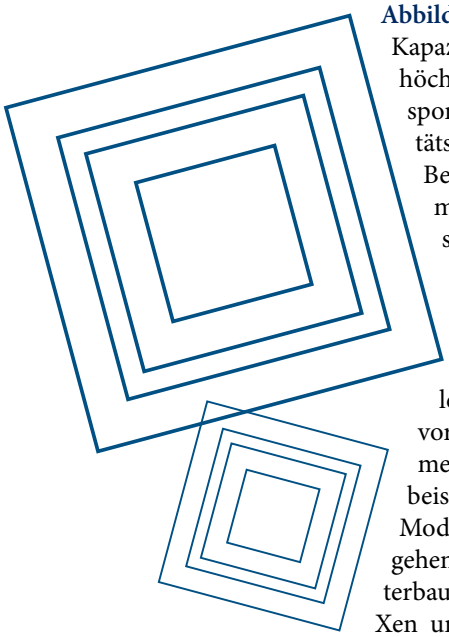


Abbildung 10: Eindritter Raum steht temporär kostenlos zur Verfügung. Platz ist reichlich, doch die Rückkehr in die Enge später umso schmerz-

Ob und wie der Scheduler Ressourcen kappt, hat entscheidenden Einfluss auf die erreichbare Perfomance unter Kontrolle des FSS. Im Beispiel der Kappung auf Basis der aktiven Shares (aus dem Pool aller Shares) entspricht etwa **Abbildung 9** der kleinsten Obergrenze der Kapazität, wogegen **Abbildung 10** mit der höchsten Obergrenze der Kapazität korrespondiert. Solche dynamischen Kapazitätsänderungen können sowohl der vom Benutzer wahrgenommenen Performance wie auch der Strategie der Ressourcen-Zuweisung abträglich sein.



Zusammengefasst: Das Verständnis des FS-Schedulers ist Voraussetzung für Kapazitätsmanagement in Echtzeit, etwas das Linux bislang nicht leisten kann. Der FSS ist die Grundvoraussetzung für Kapazitätsmanagement auf einem höheren Level, wie es beispielsweise der oben diskutierte Goal Mode verwirklicht. Außerdem – und weitgehend unbekannt – stellt der FSS den Unterbau aller Virtual Machine Manager wie Xen und VMware. Schließlich ist wichtig, dass alternative Scheduler auch neue Anforderungen an die Instrumentierung des Betriebssystems stellen.

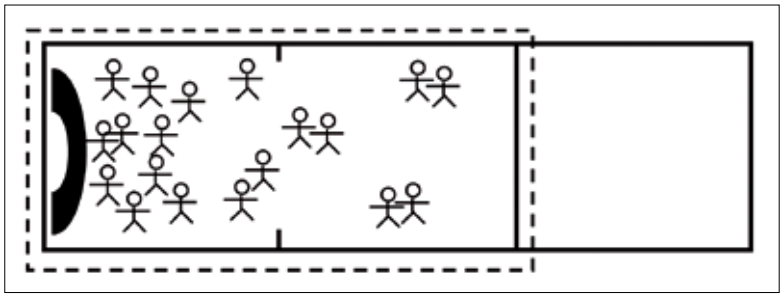


Abbildung 11: Mit zwei Räumen wurde zu viel Platz angemeitet – es kam nur die Hälfte der Gäste.

Die Herausforderung

Auf der Grundlage der vorangegangenen Diskussion lässt sich leicht vorstellen, wo nach meiner Meinung die Zukunft der Betriebssystem-Instrumentierung unter Linux liegen könnte und sollte. Sie sollte dem Vorbild der Mainframes nacheifern. Zudem: Linux ist keine Fremder für den IBM Mainframe. So läuft es etwa unter System Z in einer logischen Partition oder LPAR.
Die für Linux relevanten Mainframe-Features sind:

1. Performance Management mit einem Fair-Share-Scheduler samt einer angemessenen Instrumentierung.
2. Performance-Management mit einem Goal-Mode-Scheduler und entsprechender Instrumentierung.
3. Single Image Cluster Management.

Das erste Ziel wurde oben schon besprochen und in Form des Xen-Servers existiert sogar bereits eine Implementierung. Obgleich das zweite Ziel in Unix und Linux noch vollkommen unbekannt ist, ist der GMS doch die logische Fortsetzung der impliziten Ziele des FSS. Der GMS verlangt, dass der Administrator Performance. Ziele vorgibt. Das können Antwortzeiten oder Abarbeitungszeiten für Batch-Jobs sein. Auf dieser Grundlage kann der Scheduler dann ständig über den aktuellen Ressourcenverbrauch in Bezug auf die gesteckten Ziele Rechenschaft ablegen.
Die genaue Arbeitsweise des GMS zu beschreiben, führte hier zu weit, Es liegt aber auf der Hand, dass er nicht nur – wie der FSS – den Ressourcenverbrauch beobachten muss, sondern zugleich die Wartezeit in der Run-Queue, die direkt proportional zur Länge der Queue ist. Stellt sich dabei beispielsweise heraus, dass ein Job sein Performance-Ziel übertrifft (also zu viele CPU-Zyklen konsumiert), dann bestraft ihn der GMS, indem er ihn näher an das Ende der Run-Queue verschiebt. Dadurch braucht er nun länger, um wieder die CPU belegen zu können. Für Prozesse, die die vorgegebenen Ziele nicht erreichen, funktioniert das umgekehrt.
Natürlich ist nichts von dem umsonst. Die CPU-Zyklen müssen irgendwo her kommen und der Administrator muss die Jobs so priorisieren, dass die wichtigsten genug CPU-Zeit erhalten, um ihre Vorgaben zu erfüllen. Für jedes Ziel berechnet der Scheduler zugleich den Grad, zu dem er es erreicht – den Performance-Index.

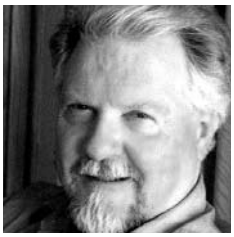
Man darf erwarten, dass IBM diese Prozesse künftig noch weiter automatisiert und sie sich selbst einstellen können.
Mit Hilfe des dritten Feature, Cluster Management, lassen sich die Ressourcen mehrerer Mainframes so zusammenfassen, dass man sie wie eine einzelne Plattform verwaltbar sind. In der IBM-Terminologie heißt eine solche Zusammenfassung mehrerer Rechner Sysplex und erfordert zusätzliche Hardware, die so genannte Coupling Facility, die als Memory Interconnect und Cache fungiert. Auf der Hardware-Ebene hat die Technik viel mit der Beowulf-Plattform gemeinsam (9). Darber hinaus aber ist der Sysplex in der Lage gleichzeitig verschiedene Applikationen auszuführen und dabei nach außen wie ein einzelnes System zu erscheinen.
Dieser Abschnitt wollte nicht vorschlagen, Linux solle die Mainframes ersetzen – selbst wenn daran nichts Falsches wäre. Stattdessen sollte er zeigen, dass es Linux heute eib gutes Systemmanagement fehlt und dass die Mainframes hier ein Vorbild sein und eine Vorstellung vermitteln können, welche Features nötig sind.

Fazit

Für Linux ergibt sich eine große Chance im Server-Markt, Ich habe versucht zu zeigen, wie sie ergriffen werden könnte. Das verursacht sicher nicht denselben Adrenalinrausch, den noch mehr Gigahertz oder noch mehr Megabytes auslösen. Die wirkliche Herausforderung besteht aber darin, durch eine verbesserte Instrumentierung ein kohärenteres Systemmanagement zu erreichen, als es gegenwärtig unter Linux vorzufinden ist. Der Artikel hat gezeigt, dass Mainframes dafür ein brauchbares Vorbild abgeben.
Während die Entwicklung kommerzieller Unix-Server durch den nötigen Return-On-Investment beschränkt ist, kann Linux relativ befreit von finanziellen Forderungen agieren. Doch mit der Freiheit geht Verantwortung einher und im Fall von Linux eine hoch organisierte Softwareentwicklung. Bis zum Beweis des Gegenteils scheint die Motivation und Kultur der Linux-Community im Ganzen einer Entwicklung, wie sie dieser Beitrag vorgeschlagen hat, entgegenzustehen, so dass die Gelegenheit möglicherweise verpasst wird. Andererseits mag auch eine Universität oder Forschungsgruppe die Aufgabe übernehmen, die nötigen organisatorischen und Forschungskapazitäten zusammenzuführen und den Prozess in die erforderliche Richtung zu lenken. Ein erfolgreiches Beispiel für eine solche Entwicklung ist das von der NASA gesponserte Beowulf Cluster-Projekt. Auch IBM unterstützt zahlreiche nicht-kommerzielle Linux-Projekte.
Eine Gruppe von Linux-Entwicklern möchten das Betriebssystem schlank und effizient halten. Der Artikel versuchte zu zeigen, wo ich dagegen eine Chance für Linux auf dem Server sehe. Vergleicht man die Desktop-Performance mit einem Sportwagen, dann beschreibt dieser Beitrag eher die Perspektive eines Sattelschleppers. Logischerweise erscheint es schwierig, beide Transportmittel in einer Code-Basis zu vereinen.
Auf jeden Fall aber besteht der dringende Bedarf für ein besseres, kohärentes Systemmanagement in der IT-Industrie, das aber wegen der proprietären Einzelinteressen kommerzieller Hersteller festzustecken scheint. In der Folge hinkt das Erreichte 10 bis 20 Jahre hinter dem her, was das Mainframe-Systemmanagement heute leistet.
Linux ist andererseits relativ frei von solchen Zwängen und deshalb bietet sich ihm die einmalige Chance, einen Ausweg aus der Sackgasse zu finden und mit Hilfe einer verbesserten Instrumentierung zum Systemmanagement des 21. Jahrhunderts aufzuschließen. *(jcb)* ■■■

Infos

(1) N. J. Gunther: Load Average enträtselt, Linux Magazin 08/2007, S. 84
(2) N. J. Gunther: Berechenbare Performance, Linux Technical Review 2, 2007, S. 112
(3) A. Kumar: Multiprocessing with the Completely Fair Scheduler (http://www.ibm.com/developerworks/linux/library/l-cfs)
(4) Systems Management: Universal Measurement Architecture: (http://www.opengroup.org/pubs/catalog/c427.htm)
(5) R. Pettit: Formalizing Performance Metrics in Linux, CMG Conference, 1999, S.p. 262
(6) N. J. Gunther: Guerrilla Capacity Planning, Springer-Verlag, 2007
(7) IBM Workload Manager: (http://www-03.ibm.com/servers/eserver/zseries/zos/wlm)
(8) Linux unter System Z: (http://www-03.ibm.com/systems/z/os/linux)
(9) Beowulf: (http://en.wikipedia.org/wiki/Beowulf%28computing%29)



Der Autor

Neil Gunther, M.Sc., Ph.D., ist ein international bekannter IT-Consultant. Er gründete 1994 die Firma Performance Dynamics Company ((www.perfdynamics.com)). Vorher bekleidete Dr. Gunther Positionen in Forschung und Management am JPL der NASA (Voyager- und Galileo- Missionen), bei Xerox PARC und Pyramid/Siemens. Heute arbeitet Dr. Gunther für Quantum Information Technologies. Er ist Mitglied von AMS, APS, ACM, CMG und IEEE.