Ansätze zum Schließen der semantischen Lücke zwischen Anwendung und Betriebssystem

Dr. H. Ritter, Dr. T. Voigt, Prof. Dr. J. Schiller (hritter|voigt|schiller@inf.fu-berlin.de)
FU Berlin, Institut für Informatik, Arbeitsgruppe Technische Informatik

1. Einleitung

Dienstgarantien und Echtzeitanforderungen in Betriebssystemen zu erfüllen, bleibt bisher immer noch zum größten Teil speziell dafür entwickelten Betriebssystemen vorbehalten. Diese Betriebssysteme, die zum Teil auf Unix/Linux basieren ([1], [2]), zum Teil aber auch eigenständige Neuentwicklungen darstellen ([3], [4], [5]), sind für den Einsatz in technischen Umgebungen gedacht, z. B. Maschinensteuerung, Fahrzeuge, Flugzeuge etc. Diese Systeme erlauben jedoch nicht, moderne verteilte Anwendungen zu betreiben, oder wenn (z.B. RT-Linux), dann ohne die Unterstützung von Dienstgarantien.

Bei Betriebssystemen wie Windows oder Linux, die zusammen mit der Java VM oder der Common-Language-Runtime (CLR) des .NET-Rahmenwerks eine mächtige Plattform für verteilte Anwendungen bieten, finden sich auf der anderen Seite zwar mittlerweile Ansätze für eine Unterstützung von Dienstgüteparametern im Kern, z. B. Verkehrsformung mit Hilfe der GQoS-API unter Windows [6] bzw. der netfilter/traffic-control-Schnittstelle unter Linux [7] oder der Möglichkeit zur Priorisierung von Prozessen in beiden Systemen von der Anwendungsebene aus. Auf der anderen Seite werden diese Möglichkeiten von bestehenden Anwendungen aber kaum systematisch genutzt.

Hier existiert eine Lücke zwischen den Anforderungen der Anwendungen und der Unterstützung durch das Betriebssystem. Das Problem liegt dabei aber nicht darin, dass lediglich eine standardisierte Schnittstelle zwischen Betriebssystem und Anwendungen fehlt, die es erlauben würde, eine Reihe von Dienstgüteanforderungen von der Anwendung aus zu spezifizieren. Vielmehr fehlen diese Schnittstellen vor allem deshalb, weil Anwendungen bzw. Anwendungsentwickler keine genaue Kenntnis darüber besitzen, wie Anforderungen, die aus der Kooperation verteilter Anwendungen resultieren, in die technischen Parameter des jeweiligen Betriebssystems umzusetzen sind.

Im Bereich der Forschung an multimedialen Endgeräten wurden daher bis heute zahlreiche Rahmenwerke geschaffen, die diese Umsetzung der Anforderungen der Anwendungen auf technische Parameter automatisieren sollten ([8], [9], [10], [11]). Hier stellen sich aus heutiger Sicht zwei Probleme: Zum einen ist das die hohe Komplexität derartiger Ansätze, die viel Rechenzeit verbraucht und den Einsatz in leistungsschwachen mobilen Endgeräten erschwert. Zum anderen beziehen die wenigsten dieser Ansätze den Benutzer mit ein, dessen Anforderungen an eine verteilte Anwendung sich aber zur Laufzeit ändern können und damit nicht vorausberechnet werden können.

In diesem Beitrag werden zunächst Ansätze vorgestellt, die eine Einbeziehung des Benutzers ermöglichen und die Einhaltung von Dienstgüteparametern unter Verzicht auf Vorausberechnung ermöglichen. Dabei zielen die vorgestellten Untersuchungen auf Standard-Betriebssysteme ohne harte Echtzeitanforderungen ab. Es wird gezeigt, dass eine zufriedenstellende Dienstgüteunterstützung vollständig ohne Wissen um die absoluten Werte der Dienstgüteparameter einer Anwendung (Datenrate, Bildrate etc.) erreicht werden kann.

Am Ende des Beitrags werden Probleme diskutiert, die durch den Einsatz von Middleware wie Java und .NET bei der Dienstgüteunterstützung entstehen, und die Gegenstand aktueller Untersuchungen der Arbeitsgruppe sind.

2 Ansatz zur Einbeziehung des Anwenders

Aus der Tatsache, dass umfangreiche Dienstgütearchitekturen nicht ihren Weg in Standard-Betriebssysteme gefunden haben, resultiert der Vorschlag einer leicht zu benutzenden und erweiterbaren Dienstgüteunterstützung unter Windows oder Linux. Zur Einbeziehung des Benutzers wurde die in Abbildung 1 dargestellte graphische Schnittstelle (die Windows-Version ist unter [12] als Installationsversion verfügbar) geschaffen:

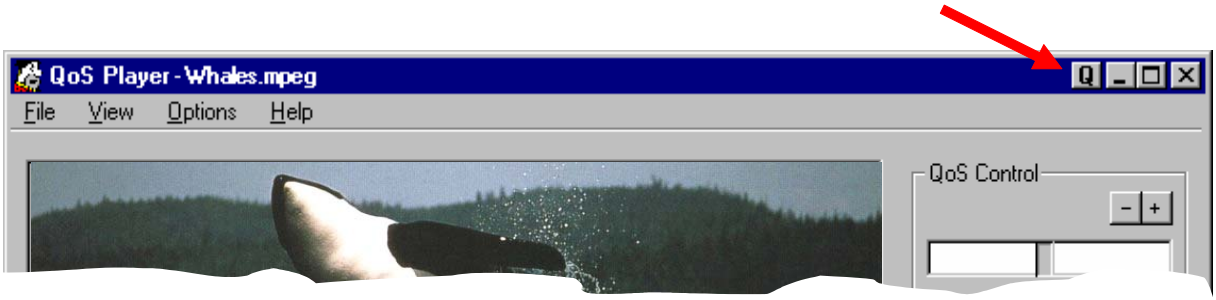

Abbildung 1: Schaltfläche zur Steuerung der Dienstgüte durch den Benutzer

Es handelt sich dabei um eine zusätzliche, mit „Q“ markierte Schaltfläche, die ohne Änderungen der Anwendungen im Fensterverwalter integriert wurde und damit allen Anwendungen zur Verfügung steht. Ein Klick mit der linken Taste der Maus auf diese Schaltfläche erzeugt ein einfaches Signal an das Betriebssystem, dass der Benutzer mit der derzeit von der dazugehörigen Anwendung gebotenen Dienstgüte unzufrieden ist, ein rechter Mausklick sendet ein entgegengesetztes Signal. Auf weitere Möglichkeiten zur Parametrisierung wurde bewusst verzichtet, um den Benutzer nicht vor eine verwirrende Auswahl ihm unbekannter technischer Parameter zu stellen.

Das erzeugte Signal bewirkt im Betriebssystemkern eine Veränderung der Gewichtungen innerhalb des Verkehrsformers und innerhalb der Prozessorzuteilung. Dabei wurde zunächst eine einfache Heuristik vorgeschlagen: Bei Anwendungen mit offenen Netzwerkverbindungen wird zunächst der Verkehrsformer benachrichtigt, ansonsten die Prozessorzuteilung. Im Folgenden soll die Wirkungsweise dieser einfachen Mechanismen an zwei Beispielen erläutert werden.

3 Kopplung mit der Prozessorzuteilung

Das Signal, das der Benutzer durch einen Klick auf die neu eingefügte Schaltfläche erzeugen kann, kann auch von einer Anwendung aus generiert werden, die dazu natürlich neu übersetzt werden muss. In dem hier vorgestellten Beispiel wurde eine einfache Multimedia-Anwendung, ein MPEG-Player der eine lokal abgelegt Datei abspielt, so verändert, dass sie immer dann, wenn die Rate der abgespielten Bilder unter einen vorgegebenen Wert fällt (in diesem Fall 20 Bilder/Sekunde), ein solches Signal generiert. Eine solche Anwendung wurde als proaktiv bezeichnet, da sie im Gegensatz zu klassischen adaptiven Anwendungen bei Ressourcenmangel nicht die gebotene Leistung reduziert (adaptives Verhalten) sondern nach außen hin aktiv wird, indem sie mehr Ressourcen anfordert.

In Abbildung 2 ist das Ergebnis zweier Messreihen überlagert dargestellt. Im Intervall zwischen t=15 s und t=30 s wurde durch eine rechenintensive weitere Anwendung eine Ressourcenknappheit herbeigeführt. Während die klassische adaptive Anwendung mit einer Reduktion der ausgespielten Bildrate reagiert, ist deutlich zu sehen, dass die proaktive Anwendung nach kurzer Zeit wieder über die Rate von 20 Bildern/Sekunde gelangt.

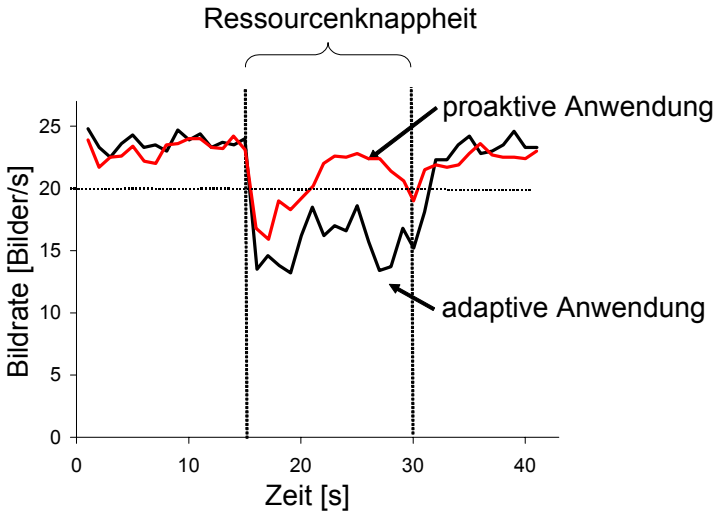

Abbildung 2: Verbesserung der Dienstgüte durch ein einfaches Signal der Anwendung

Das gleiche Verhalten ergibt sich, wenn die Anwendung nicht von sich aus aktiv wird, sondern der Benutzer bei Feststellen einer Verschlechterung der Dienstgüte die im vorigen Absatz beschriebene Schaltfläche aktiviert.

4 Dynamisches Gleichgewicht bei der Paketzuteilung

Das Prinzip, dass ein einfaches Signal ausreicht, um Dienstgüteanforderungen gerecht zu werden, wurde auch bei der Zuteilung der Datenrate angewandt. Dazu wurde auf Basis der vorhandenen Einstiegspunkte für Paketzuteilung im Linux Betriebssystem ein Verkehrsformer realisiert, der anforderungsgesteuert die verfügbare, konstante Gesamtdatenrate in Ausgangsrichtung auf konkurrierende Anwendungen verteilt. Die Datenrate wird zunächst gleichmäßig auf alle Anwendungen verteilt, da ja keine explizite Rateninformation vorliegt.

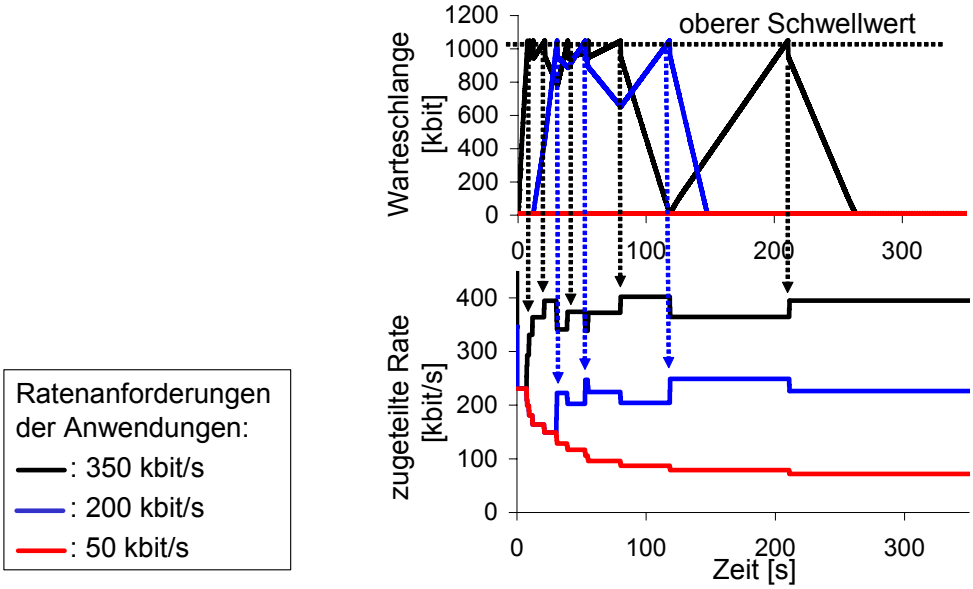

Abbildung 3: Verteilung der ausgehenden Datenrate ohne explizite Ratenanforderungung

Mit Hilfe einer Ausgangswarteschlange stellen nun proaktive Anwendungen fest, dass die zunächst zur Verfügung gestellte Datenrate nicht ausreicht und generieren ein Signal an das Betriebssystem, sobald die Warteschlange überzulaufen droht. Daraufhin weist der Verkehrsformer dieser Anwendung einen höheren Anteil an der Gesamtdatenrate zu, und den anderen Anwendungen ihrem Anteil entsprechend weniger. Bei mehreren proaktiven Anwendungen wer-

den anschließend weitere Anwendungen einen Ressourcenmangel signalisieren, das System schwingt sich erst nach einer gewissen Zeit ein. Da immer wieder Anwendungen beendet und andere neu gestartet werden, wird das Gleichgewicht immer wieder neu hergestellt; erst dies ermöglicht auch die Einbeziehung von nicht vorhersehbarer Benutzerinteraktion.

In Abbildung 3 ist das Ergebnis einer Simulation des Verfahrens mittels des Simulationswerkzeugs OMNeT++ mit drei Anwendungen wiedergegeben, die jeweils über eine Ausgangswarteschlange der Größe 1024 kbit verfügen. Sobald die Ausgangswarteschlange diesen Wert erreicht, genereriert die jeweilige Anwendung ein Signal an das Betriebssystem. Die Rate, mit der die Anwendungen zu senden versuchen, beträgt 350 kbit/s, 200 kbit/s und 50 kbit/s, die zur Verfügung stehende Gesamtdatenrate beträgt 691,2 kbit/s. Es ist sehr gut zu sehen, dass sich nach kurzer Zeit ein stabiler Zustand einstellt, der den Anforderungen aller drei Anwendungen gerecht wird. Weitere Untersuchungen haben gezeigt, dass bei nicht ausreichender Gesamtdatenrate zwar die Zuteilung leicht schwankt, aber die Verteilung der Gesamtdatenrate auf die einzelnen Anwendungen dem Verhältnis ihrer benötigten Datenraten entspricht. Dabei ist zu beachten, dass auch hier wiederum auf die Weitergabe expliziter Ratenangaben von der Anwendung an das Betriebssystem oder eine dritte Instanz, beispielsweise einen Dienstgütemonitor, verzichtet werden konnte.

5 Probleme mit Middleware

Der vorgestellte Ansatz wurde sowohl simulativ als auch unter den Betriebssystemen Linux und Windows untersucht. Dabei wurde jedoch ausgenutzt, dass zwischen dem Betriebssystem und den Anwendungen eine Zuordnung dann vergleichsweise leicht fällt, wenn sich keine Middleware dazwischen befindet. Dadurch können Anwendungen, die in einem Fenster laufen, eindeutig einem Prozess im Betriebssystem-Kern zugeordnet werden, ebenso können Datenströme sehr leicht zugeordnet werden.

Durch eine Middleware wie die Java VM oder die Common Language Runtime von .NET werden diese Zuordnungen sehr erschwert. Schon jetzt werden Zuordnungen dadurch erschwert, dass viele verschiedene Anwendungen innerhalb eines Prozesskontexts gestartet werden können, beispielsweise innerhalb eines Internet Browsers wie dem Internet Explorer. Hier ist ein Ausweichen auf threads eventuell noch möglich. Wenn jedoch als Urheber eines Datenstroms oder einer Anforderung nicht mehr eine für den Anwender sichtbare Anwendung bestimmt werden kann, da die Kommunikation mit dem Betriebssystem einzig über die Middleware stattfindet, entspricht dies einer weitgehenden Kapselung des Betriebssystems, die semantische Lücke zwischen Betriebssystem und Anwendungen wird größer.

Das .NET-Rahmenwerk steht noch am Anfang der Entwicklung, zudem ist der Quellcode einer nicht-optimierten, aber voll funktionsfähigen Version der CLR offengelegt. Das gibt die Chance, zu untersuchen, welche Möglichkeiten es gibt, die durch die Middleware vergrößerte Lücke zwischen Betriebssystem und Anwendungen zu schließen. Dabei sollte es sich als vorteilhaft erweisen, dass keine komplexen Parametersätze unterstützt und weitergegeben werden müssen. Allerdings müssen zusätzlich Zusammengehörigkeiten innerhalb der CLR bei der Dienstgüteunterstützung berücksichtigt werden, wenn beispielsweise die Leistungsfähigkeit einer Anwendung nicht nur von dem thread der Anwendung, sondern auch von threads, die von der CLR aus gestartet werden, abhängt. Hier soll untersucht werden, wie solche Abhängigkeiten erfasst und dargestellt werden können, um dann entscheiden zu können, auf welcher Ebene des Gesamtsystems diese Zusammenhänge berücksichtigt werden können und müssen.

Der praktische Teil der Arbeiten soll unter Windows CE erfolgen, sobald der Quellcode des Netzwerkteils von Windows CE, wie von Microsoft angekündigt, an der FU Berlin verfügbar ist.

Literatur

[1] Barabanov, Yodaiken: Real-Time Linux. Linux Journal, März 1996

[2] Liedtke, Dannowski, Elphinstone, Liefländer, Skoglund, Uhlig, Ceelen, Haeberlen, Völp: The L4Ka Vision. Verfügbar unter: http://l4ka.org

[3] EUROSplus. EUROS Embedded Systems GmbH, Informationen unter http://www.kaneff.de

[4] VxWorks. Windriver Inc. Informationen unter http://www.windriver.com

[5] pSOS. Windriver Inc., Informationen unter http://www.windriver.com

[6] Isemberger: Windows 2000 Quality of Service. New Riders, 1999

[7] Wehrle, Pählke, Ritter, Müller, Bechler: Linux Netzwerkarchitektur. Addison-Wesley, 2002

[8] Nahrstedt, Smith: The QoS Broker. IEEE Multimedia, 2(1), 1995

[9] Nahrstedt, Smith: Design, Implementation and Experiences of the OMEGA End-Point Architecture. IEEE JSAC, Special Issue on Distributed Multimedia Systems and Technology, 14(7), 2000

[10] Campbell, Coulson, Hutchison: A Multimedia Enhanced Transport Service in a Quality of Service Architecture. In: Proceedings of 4th International Network and Operating System Support for Digital Audio and Video, 1993

[11] Gopalakrishnan, Parulkar: Framework for QoS Guarantees for Multimedia Applications within an Endsystem. Swiss-German Computer Society Conference, 1995

[12] Q-Button Installationsversion, verfügbar unter: http://www.inf.fu-berlin.de/~hritter/windows/index.html