Seminararbeit

CONTENT SERVER

Johannes Kohlmann
23.11.2004

Seminar
Content Delivery Networks
WS 2004/05

Universität Mannheim
Lehrstuhl Praktische Informatik IV: Prof. Dr. Wolfgang Effelsberg
Betreuer: Dipl.-Ing. Hans Christian Liebig

Inhalt

1. Einleitung

Dass es im Internet Server, die etwas anbieten, und Clients, die auf die Server zugreifen, gibt, ist im Prinzip nichts Neues. Aber was passiert, wenn plötzlich tausende Clients gleichzeitig auf einen Server (Beispiel CNN) zugreifen, Webseiten lesen möchten und beispielsweise die aktuellen Nachrichtenbeiträge auf Video anschauen möchten? Das kann jeder Server nur bis zu einer maximalen Anzahl Clients, danach bricht dieser wegen Überforderung zusammen, so dass kein Client mehr was von ihm hat. Oder die Festplatte oder die Datei, auf dem der aktuelle Fernsehbeitrag gespeichert ist, wird durch das gleichzeitige Lesen der gleichen Datei an verschiedenen Stellen gar zerstört.
Kann man diese Probleme lösen? Man könnte einen leistungsstärkeren teureren Server einsetzen – doch der ist dann auch – nur später - an seiner Leistungsgrenze. Also muss vielleicht die Logistik eines „Servers“ so überarbeitet werden, dass jeder Client bedient werden kann.

Die erste Möglichkeit ist, mehrere Server zur Last- und Ressourcenverteilung aufzustellen und ein Routingsystem einzurichten. Solche Anlagen nennt man auch Serverfarms, die eine Größe zwischen zwei und mehreren hunderten an Servern haben können. Bestimmte Module verteilen die Requests an die einzelnen Server, verteilen die Serverressourcen sinnvoll, oder managen ein verteiltes Dateisystem.

So wird primär in Webcontent und Multimediacontent unterschieden, da für beide unterschiedliche Infrastrukturen sinnvoll sind.

Bei einem Webcontent-Server kann ein Ressourcenverteilungssystem wichtigen Seiten wie beispielsweise der Startseite höhere Priorität und Unterseiten, die selten besucht werden, eine niedrige Priorität zuordnen. Nach der Prioritätenliste werden dann die Ressourcen des Servers dementsprechend verteilt. So kann die Verfügbarkeit des Servers gesteigert werden.

Sollen multimediale Objekte mit Streams angeboten werden, schaffen bestimmte Algorithmen und Methoden für das Videostreaming eine Möglichkeit, dass viel mehr Clients diesen Service in Anspruch nehmen kann, ohne dass ein Server überlastet wird und damit seine Arbeit einstellt oder dass die Festplatten-Leseköpfe zu sehr beansprucht werden.

Es gibt unzählige Lösungen und Ansätze, wovon hier einige im Zusammenhang vorgestellt werden, um einen kleinen Einblick in die Welt eines Content-Servers zu geben.

2. Herausforderung an Content-Server

Server müssen so gut konzipiert werden, dass sie auch im worst-case noch alle Clients bedienen können. Ein einfacher Server kann dies nicht mehr bewerkstelligen. Es wurde in Vergangenheit, aber vor allem auch heute, Lösungen für das expandierende Internet gesucht. Es wer-

den an allen „Enden“, also sowohl bei Hard- und Software des Servers, bei dem Management des Arbeitsspeichers, der Festplatten, bei der Netzwerkstruktur, den Routern, den phyikalischen Gegebenheiten der Netzwerkkabel und auch auf der Clientseite nach besseren Lösungen geforscht.
Bei Webseiten, vor allem bei dynamischen Inhalten und bei Streams, bei denen der Server einiges mehr an Ressourcen benötigt, ist eine Lastverteilung auf mehreren Servern nötig. Dazu verteilt das Admission Control die Requests auf allen vorhandenen Server, das Load Balancing verteilt die Arbeitsvorgänge. Hier kann es zu keinem Serverzusammenbruch kommen, da wenn alle Server Ihre Grenze erreicht haben, das Admission Control weitere Anfragen ablehnt. Dies sieht bei dem Quality Of Service anders aus. Hier werden zwar einzelne Webseiten bzw Objekte verschieden gewichtet, so dass beispielsweise Startseiten und Streams höhere Priorität haben und im Netzwerk auch schneller ausgeliefert werden, als die anderen Objekte, doch hier kann es trotzdem zu einem Zusammenbruch kommen, wenn beispielsweise plötzlich zu viele gleich-gewichtete Objekte abgerufen werden (bsp. Startseite). Bei Streams kann es auch zu einer Verstopfung des Internets oder des Netzwerks führen, da nieder-gewichtete Objekte schon gar keine Chance mehr haben, wenn zu viele hoch-gewichtete Objekte ausgeliefert werden sollen.
Die hier vorgestellte Idee „DALA“ ist das dynamische und automatische Anpassen der Serveranzahl in Abhängigkeit der Nachfrage von beispielsweisen Multimedia-Streams. Testergebnisse zeigen eine hohe Effizienz und eine hohe Anzahl an Clients, die auf dieses System zugreifen können, auf. Falls es zur Situation kommen sollte, dass alle abgespeicherten Streams eine gleich große Nachfrage haben, werden plötzlich alle Server mit allen Stream-Requests gleichzeitig belastet, was im Extremfall zu einem Zusammenbruch bzw. zur Ablehnung aller Anfragen führen könnte.
Bei der Idee des Gatekeepers als Proxyrechner zwischen Applikation und Datenbank werden die Requests mit einem interessanten Algorithmus verarbeitet. Hier werden die Requests in eine Warteschlange übertragen, nach ihrer Ausführungsdauer sortiert, und dann möglichst viele kurze Requests gleichzeitig ausgeführt in Abhängigkeit von der maximalen Leistung der Server. Dass die Requests mit der kürzesten Ausführungsdauer zuerst ausgeführt werden, bietet den Vorteil, dass möglichst viele Requests in nur kurzer Zeit fertig sind, und so Wartezeiten minimiert werden können. Doch dies funktioniert nicht mehr, sobald die Ausführungsdauer der Requests alle gleich groß sind. Hier wird dann sequentiell abgearbeitet, was auch ggf. ziemlich lange dauern kann.
Bei Multimedia-Servern gibt es verschiedene Ansätze, wie möglichst Speicher-, Netzwerk-, und Ressourceneffizient Anfragen für mehrere Clients gleichzeitig behandelt werden können. Die interessanteste und in mehreren Tests bestätigte effiziente BubbleUp bietet in vielen Hinsichten eine gute Lösung für einen Streamingserver. Es hat zwar relativ lange Zugriffszeiten und den Nachteil, dass die Dauer der Zugriffe nicht selbst bestimmt werden kann, doch dies wird mit dem guten Speichermanagement wieder wett gemacht.

Theoretisch wäre es das Optimum, alle bisher entwickelten Lösungen von allen “Enden“ des Content-Servers gleichzeitig zu benutzen, so dass die physikalisch maximal-erreichbare Anzahl an Clients, die gleichzeitig bedient werden müssen, erreicht werden kann. Doch dies ist in der Praxis leider nicht möglich, da jede Lösung ihren Eigenanteil an Ressourcennutzung benötigt und zu viele Problemstellen in das Gefüge einbringt, so dass mehrere Lösungen eher kontraproduktiv zusammenarbeiten.

3. Web-Content-Server

Hier werden jetzt Lösungen für einen Web-Content-Server vorgestellt. Dazu zählt hier der Quality of Service [5], Admission Control und Load Balancing [12], DALA [1] und der Proxyserver Gatekeeper [3].

3.1. Quality of Service (QoS)

Eine große Effizienz ergibt sich, wenn die Seiten eines Webservers mit Prioritäten behaftet werden, damit nicht jeder Client den gleichen Anteil an den Serverressourcen hat [5]. Das macht dann Sinn, wenn beispielsweise die Startseite zu 100% verfügbar sein soll, selten besuchte Unterseiten nicht immer verfügbar sein müssen. Mit einer solchen Verteilung können viel mehr Clients bedient werden, ohne dass die Ressourcen des Servers zu schnell zu klein werden.

Beim Quality of Service werden IP-Datenpakete mit bestimmten Merkmalen priorisiert. Mit diesem Mechanismus ist es beispielsweise möglich, VoiceOverIP oder auch Streams höhere Priorität zu geben, so dass diese Pakete schneller zum Ziel geroutet werden, als Dateidownloads und Webseiten.
Das Endgerät handelt dabei mit dem Dienstleister bei Verbindungsaufbau eine garantierte QoS-Klasse aus.
Als Beispiel gibt es hier: Constant Bitrate (CBR), realtime Variabel Bitrate (rtVBR), nonrealtime Variabel Bitrate(nrtVBR), Unspecified Bitrate (UBR) und Available Bitrate (ABR).
Quality of Service kann man mit Jitter, Fehlerrate, Verzögerung, Bitrate und Paketverlust parametrisieren.

3.2. Klassisch: Load Balancing und Admission Control

Wie in Bild 1 zu sehen ist, besteht ein solches System aus mehreren Webswitches und mehreren Webservern [12]. Hierbei nehmen die Webswitches die Client-Requests an und leiten sie an einen ganz bestimmten Webserver weiter, so dass die Last verteilt werden kann. Dabei müssen die Client-Requests aber unterschieden werden. Denn wenn eine Session mit einem Server schon besteht, muss der Request an den gleichen Server geleitet werden, bei dem auch die Session begann, da sonst Daten verloren gehen. Ein Beispiel wäre, wenn man sich auf einer Webseite mit seinem Usernamen angemeldet hat, was bei vielen Webseiten wie Foren, Shops u.s.w. nötig ist, und dann plötzlich von einem anderen Server weiterbedient wird, gehen die Sessiondaten wie der des Usernames verloren. Als User kommt man hier wieder auf die Startseite zum einloggen.
Admission Control und Load Balancing sind Funktionen zur Lastverteilung, die entweder softwaretechnisch auf Seite des Servers zwischen dem Netzwerkinterface und dem Webserverdienst eingefügt wird, oder Hardwaretechnisch im Webswitch integriert ist. Hierbei gibt es für beide verschiedene Algorithmen, die eingesetzt werden können. Jede dieser Algorithmen hat Vor- und Nachteile, die bei der Planung der Serverfarm gut erörtert werden müssen. Beispiel: Request ablehnen oder halten in einer Warteschlange, wenn alle Server überlastet sind.

Das Admission Control entscheidet dabei, zu welchem Server ein Request weitergeleitet wird. Eine Möglichkeit besteht, wenn das Admission Control die aktuellen System-Load-Werte eines jeden Servers kennt. Kann wegen Überlastung kein Server gefunden werden, an den ein neuer Request weitergeleitet werden kann, wird entweder der Request abgelehnt, oder er wird zu einem Load-Balancing-System weitergeschickt. Ein Load-Balacing-System ermittelt dabei anhand der gleichen System-Load-Werte die Server, die dann diesen Request (jetzt oder später) bearbeiten werden. Load-Balancing kann auch Requests auf mehrere Server aufsplitten, falls nötig. Dabei werden die System-Load-Werte so berücksichtigt, dass kein Server überlastet wird.
Nur durch das Load-Balancing können keine schnellen Antworten des Servers an den Client garantiert werden. Die kurze Antwortzeit des Servers realisiert primär das Admission Control durch eine möglichst optimale Verteilung der Client-Requests.
Viele Webserver haben aber nur Load-Balancing integriert, aber kein Admission Control.
Aus Kosten- und Performance-Gründen kommt Admission Control eher bei größeren Serverfarms zum Einsatz.
Besucht ein User bereits Webseiten auf dem Webserver und kommt ein neuer User auf die Webseite hinzu, so weist Admission Control oft diesem eine niedrigere Priorität zu als dem „älteren“ User. Damit können die Wartezeiten beim Wechseln der einzelnen Webseiten minimiert werden, damit ein Qualitätsstandard sichergestellt werden kann, der den User in seinem Geschwindigkeitserwartungen zufriedenstellt.

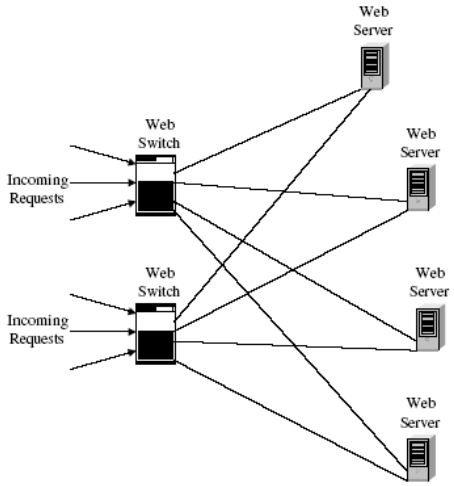

Bild 1: Webserver-System mit Webswitch. Quelle: [12]

3.3. Multimedia-Serverfarms mit: A Demand Adaptive and Locality Aware (DALA)

Ein nennenswerter Vorteil dieses Verfahrens ist das dynamische und automatische Anpassen der Serveranzahl nach der Nachfrage für ein Multimedia-Stream, der hier von den Servern angeboten wird. So kann garantiert werden, dass kein Server mit den Client-Requests für ein Stream überlastet wird. [1]

3.3.1. System Model

Es gibt N Server, die alle mit einem Hochgeschwindigkeits-Switch verbunden sind. Jeder Server hat eine eigene Festplatte für die Streaming-Inhalte. Auf Multimedia-Servern werden diese Inhalte oft als Objekte bezeichnet. Die Architektur ist nun so konzipiert, dass sowohl ein Server alle Objekte auf der einen lokalen Festplatte anbieten kann, als auch dass viele Server das gleiche Objekt anbieten können. Wieviele Server für ein Objekt genau gebraucht werden, hängt von der Anzahl der Clients ab, die dieses Objekt beziehen möchten. Je mehr Clients, desto mehr Server werden hier gebraucht, dazu im nächsten Absatz mehr.
Es werden mehrere Server zu Gruppen zusammengefasst. Auf allen Servern einer Gruppe wird das gleiche Objekt gespeichert. Eine Gruppe muss mindestens aus einem Server bestehen, und zwar aus dem, der die Originaldatei des Objektes beherbergt.
Deswegen hat die lokale Festplatte eines Servers zwei Partitionen. Auf der *Primary Copy Partition* ist die Originaldatei eines Objektes permanent gespeichert. Auf der *Dynamic service partition* werden Kopien aller Objekte, die sich innerhalb einer Gruppe befinden, gespeichert.

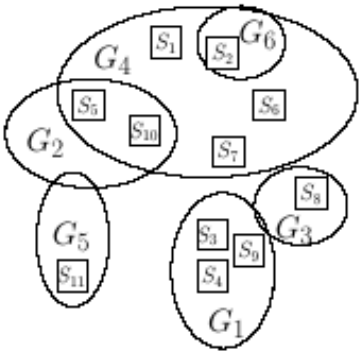

Bild 2: Einteilung der Server in Gruppen

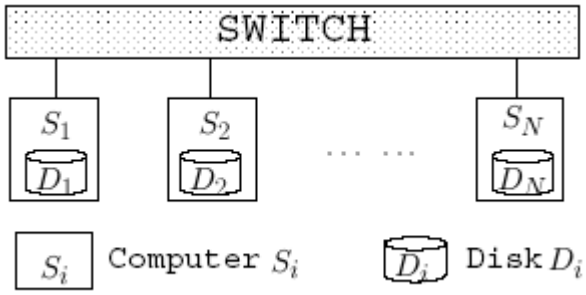

Bild 3: Infrastruktur bei DALA

Quelle: [1]

3.3.2. DALA Demand Dissemination Protocol

Jetzt kommt ein Stream-Request für ein Objekt auf einem Server in der Gruppe des Objektes an. Jeder Server besitzt ein Token-Holder, der das Admission-Control verrichtet. Der Token-Holder speichert eine Taelle mit allen Servern der Gruppe, die bei Bedarf aktualisiert wird. Dafür gibt es verschiedene Algorithmen, die verwendet werden können. Das Token-Holder ermittelt bei dem Server die Netzwerk- und die Festplattenauslastung und den noch freien Speicher-Buffer. Sind genug Ressourcen frei, wird der Request auf diesem Server angenommen und bedient. Sollte dieser Server schon ausgelastet sein, wird der Request an den nächsten Token-Holder der Gruppe weitergeleitet (meist über ein Layer-7-Switch). Sollte der Toking-Passing-Algorithmus keinen freien Server finden, heißt das, dass alle Server der Gruppe ausgelastet sind und ein weiteres Mitglied in dieser Gruppe benötigt wird. Dies wird mit dem Dynamic-Group-Membership-Protocol erreicht. Dieses kann die Größe der Gruppe ändern, indem es einen anderen (freien) Server mit in die Gruppe nimmt. Dazu im nächsten Absatz mehr. Diesem neuen Server wird dann der Request weitergeleitet. Sollte der Fall eintreten,

dass auch kein (freier) Server zum Eintreten in die Gruppe gefunden wird, muss der Request abgelehnt werden.

3.3.3. Dynamisches Anpassen der Gruppengröße

Wurde dem Dynamic-Group-Membership-Protocol mitgeteilt, dass ein neuer Server benötigt wird, sucht der aktuelle Token-Holder (vom letzten überlasteten Server) einen Rechner mit der wenigsten Überlastung aus. Hierbei bevorzugt er den Server, der schon eine Kopie des Objektes auf seiner Dynamic-Service-Parition hat. Dieser Server wird mit in die Gruppe genommen. Falls kein entsprechender Rechner mit einer bereits vorhandenen Kopie zur Verfügung steht, wird ein Server ohne Kopie des Objektes ausgewählt. In diesem Fall schickt er aktuelle Token-Holder ihm die entsprechende Kopie zu. Dafür wird auch das Hochgeschwindigkeitsnetzwerk zwischen den Servern gebraucht.
Anschließend wird allen Servern der Gruppe das neue Mitglied mitgeteilt. Mit einer Timeout-Strategie werden Server aus Gruppen wieder automatisch herausgenommen, wenn der Token-Holder des Servers zu lange keine Requests mehr bekommen hat, weil beispielsweise die Popularität des Objektes abgenommen hat. Auch dieses wird allen Gruppenmitgliedern mitgeteilt.
Somit wird anhand der Popularität der Objekte die Gruppengröße dynamisch angepasst und Requests von Clients dementsprechend an viele Server verteilt.

3.4. Transparenter Proxy mit Admission Control & Request Scheduling: Gatekeeper

Dieser transparente Proxy ist speziell für dynamischen Content vor allem im E-Commerce-Bereich angedacht [3]. Es können Servlets für die Anwendung verwendet werden.
Zu Anfang wird der Wert für System-Load auf Null initialisiert. Wenn beispielsweise ein Servlet einen Datenbankrequest erhält, wird die Arbeitsauslastung für diese Anfrage ermittelt. Wenn diese den System-Load nicht überschreitet, darf das Servlet den Request bedienen, und der ermittelte Wert wird beim System-Load dazugerechnet. Im anderen Fall wird der Request in eine Admission-Warteschlange in FIFO-Reihenfolge geschickt, damit es später ausgeführt werden kann. Wenn der Server keine Aufträge mehr hat, dekrementiert Gatekeeper den System-Load dementsprechend, so dass evtl anstehende Requests aus der Admission-Warteschlange nach und nach weitergereicht werden. Gatekeeper blockt keine Requests ab, sondern stellt sie alle in die Warteschlange. Wenn das System permanent überlastet ist, bleiben die Requests evtl sogar unendlich lang in der Schlange, so dass ein Timeout anzeigt, dass das System aufgerüstet werden sollte. Der Systemadministrator kann aber auch andere Admission-Control-Policies einrichten – wie zum Beispiel das Droppen von Requests, wenn das System überlastet ist.
Gatekeeper benutzt zudem ein Request-Scheduling, um die Antwort-Zeiten von dynamischen Webcontent zu verringern. Dazu werden die Requests, sortiert nach ihrem ermittelten Auslastungswert in die Admission-Queue geschickt. So wird erreicht, dass die Requests, welche am wenigsten Auslastung benötigen, zuerst bedient werden. Wenn der Server dann keine Jobs hat, werden ihm so viele Requests vom Anfang der Warteschlange übermittelt, bis der Wert für das System-Load der Summe der Auslastungswerte der übermittelten Requests entspricht. So kann gewährleistet werden, dass möglichst viele Requests in einem Zug bedient werden

können. Diese Vorgehensweise funktioniert allerdings nur dann zeitersparend, wenn die ermittelten Auslastungswerte der Requests nicht alle gleich groß sind.

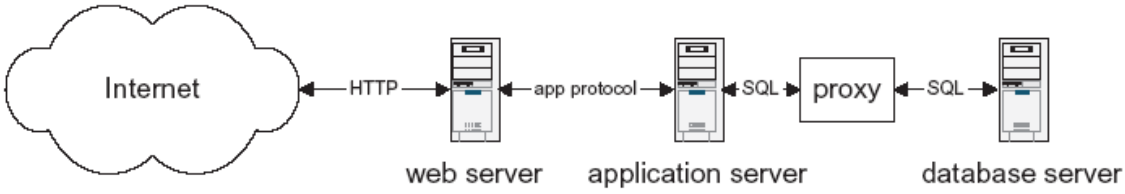

Bild 4: Platzierung des Gatekeeper-Proxy in einer E-Commerce-Infrastruktur. Quelle: [3]

4. Multimedia-Content-Server

Auf Servern mit Multimedia-Inhalten spielt sowohl die Infrastruktur als auch das Verfahren zum Streamen eine große Rolle. Nur durch ausgeklügeltes Zusammenarbeiten mehrerer Komponenten können Streaming-Server hochverfügbar gemacht werden. Generell werden Multimedia-Server mit externen Storage-Systemen mit Dutzenden an Festplatten betrieben. RAID-Lösungen sind auch geeignet. Ob es nun einen Server oder gar eine ganze Serverfarm ist, die den Streaming-Service anbieten soll, sei hier offengelassen. Wir stellen uns ein System mit mehreren Festplatten vor. Es werden hier im Folgenden mehrere Modelle und Lösungen zum eigentlichen Streamen vorgestellt, die mit enormer Ressourcenbelastung trotzdem lauffähig sein müssen.

4.1. Data-Striping

Beim Data-Striping [6] ist jedes Multimedia-Objekt wie beispielsweise ein Film in mehrere gleich große Teile (Stripes) zerteilt. Diese Stripes werden nacheinander ringsum auf allen vorhandenen Festplatten verteilt.
Wenn ein Client den Film als Stream empfangen möchte, wird im ersten Schritt der erste Stripe von der Festplatte gelesen, im zweiten Schritt wird gleichzeitig der erste Stripe versendet und der zweite von der Festplatte gelesen, im dritten Schritt wird der zweite Stripe gesendet und der dritte von der Festplatte gelesen u.s.w. bis der letzte Stripe erreicht wurde oder der Client den Empfang abgebrochen hat. (Bild 5)

Es wird also in Folge immer gleichzeitig der letzte Stripe gesendet, während der nächste schon von der Festplatte gelesen wird. In einer bestimmten Zeitspanne kann nur maximal eine bestimmte Anzahl an Stripe-Blöcken von jeder Festplatte gelesen werden. Dies ist bedingt durch die Dauer jedes einzelnen Schrittes beim Lesen und Senden eines Streams. Die maximale Anzahl von Stripe-Blöcken je Zeiteinheit wird ausschließlich von den Festplattenparametern (Lesegeschwindigkeit etc) bestimmt. Dadurch wird auch die Anzahl an Streams, die jede Festplatte zur Verfügung stellen kann, begrenzt.

Das System benötigt zudem ein Admission Control, die weitere Requests für Streams entweder zulässt oder abblockt – und zwar in Abhängigkeit der Systemauslastung. Wenn der Request zugelassen wird – also genug Ressourcen vorhanden sind -, muss das System erst warten, bis in dem Lesekreislauf die entsprechenden Festplatte mit dem ersten Stripe erreicht wurde. Erst dann beginnt der Stream nach gleicher Weise wie oben.

Diese Vorgehensweise ist für Objekte mit konstanter Bit-rate (CBR) optimiert. Für Objekte mit variabler Bit-rate (VBR) ist es nicht ganz so optimal.

Was die Performance angeht, ist es oft üblich, dass wenn neue Objekte auf den Server kommen, dafür andere alte Objekte gelöscht werden. Um Fragmentierung der Festplatten zu vermeiden, sind die Größen der Datenblöcke vorgeschrieben und werden dementsprechend reserviert.

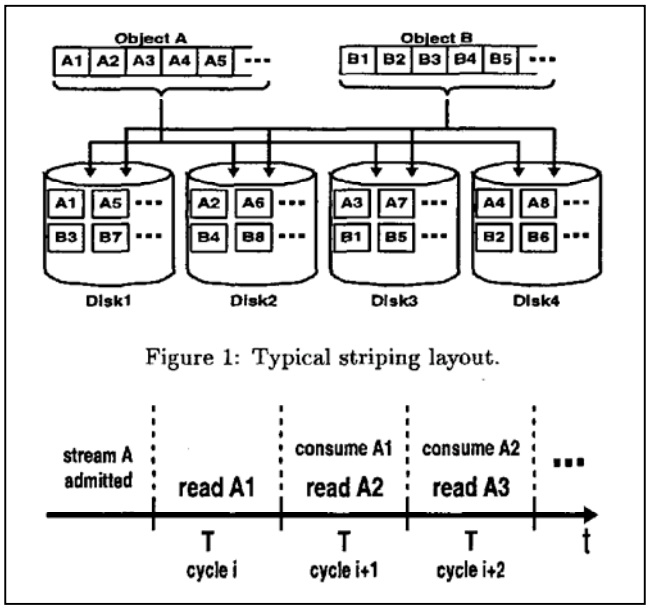

Bild 5: Verteilung und Ablauf beim Data-Striping Quelle: [6]

4.2. Random Data Allocation (RDA) / Randomized I/O (RIO)

Ein Multimedia-Objekt wird in mehrere gleich große Blöcke zerteilt, bei der jeder Block an zufällig ausgewählten Positionen auf einer zufällig ausgewählten Festplatte gespeichert wird. Von einem Bruchteil von zufällig ausgewählten Blöcken im System können Kopien angelegt werden. Sie werden an verschiedenen Stellen gespeichert, solange es nicht die per Zufall ausgewählte Festplatte ist [6][16]

4.3. Speichermanagement mit Zyklusbasiertem Scheduling

Um eine effiziente Ausnutzung der Bandbreite für Stream-Requests zu erreichen, bietet es sich an, das ganze in Zyklen und Zeitperioden zu verarbeiten [9]. Dabei wird in der einen Zeitperiode sowohl das in der letzten Gelesene über das Netzwerk geschickt, als auch der nächste Block eines anderen Streams gelesen. Das geht immer so weiter in einer Art Kreislauf, es wird ein Block gelesen und in der nächsten Zeitperiode verschickt. Diese Funktionsweise lässt den vorgesehenen Speicher in einem begrenztem Rahmen. Hier gibt es unter an-

derem zwei bekanntere Ansätze, die auf diesen Zyklusbasiertem Scheduling aufbauen. Das Sweep- [17] und das Fixed-Stretch-Verfahren [8][17].

4.3.1. Sweep (Fahrstuhl-Prinzip)

Das Sweep-Verfahren [17] zielt darauf ab, die Kosten der Suchvorgänge möglichst minimal zu halten. Dazu wird nach und nach ein Segment von jedem Stream gelesen, und ebenso nacheinander in den Buffer geschrieben. Von dort wird ein Segment nach dem anderen an die Clients über das Netzwerkinterface verschickt. Durch das serielle Arbeiten ist die IO-Zeit-Regulierbarkeit sehr hoch, denn die nächsten Segmente werden erst verarbeitet, wenn das aktuelle Segment fertig ist. Die Zugriffszeit (Latency) ist minimal, aber das ungünstige Speicher-Sharing ist hier als wesentlicher Nachteil zu nennen.

4.3.2. Fixed-Stretch

Es müssen also – wie bereits beschrieben - von jedem Stream Daten gelesen, und daraufhin gesendet werden. Um den Zeitaufwand zu optimieren, werden Requests beim Fixed-Stretch-Verfahren [8][17] in Zyklen (Zeit T), wie gerade im Kapitel „Zyklusbasiertes Scheduling“ [9] beschrieben, verarbeitet. In einem solchen Zyklus liest der Server ein Segment von jedem Stream-Request, jedoch maximal N Stück. Ein solcher Zyklus wird unterteilt in N Blöcke, genannt Service-Slots, also für jeden Stream ist ein Slot vorgesehen. Diese werden dann in einem Zyklus nacheinander verarbeitet.
$\gamma(CYL)$ sei hier der worst-case Seek Delay. Jedes Segment sei S Bytes lang.

Es findet folgender Ablauf statt:

1. Am Anfang eines Slots wird der End-Of-Slot-Timer auf einen Wert Δ gesetzt.
2. Ist hier keine Anfrage für einen Stream zu verarbeiten, springe zu Punkt 6
3. Alloziiere S Speicher für die Verarbeitung der Anfrage in diesem Slot.
4. Setze den IO-Timer auf $\gamma(CYL)$, und starte das Lesen von der Festplatte. Wenn der IO-Timer abgelaufen ist, wird der Datentransfer begonnen. Dieser Punkt wird Playback-Point bezeichnet.
5. Der Datentransfer schaufelt Daten in den Buffer und der benutzte Speicher kann somit (für diesen Slot) verkleinert werden.
6. Wenn der End-Of-Slot-Timer abgelaufen ist, muss der Datentransfer abgeschlossen sein, ansonsten wird er an dieser Stelle abgebrochen. Jetzt wird im nächsten Slot am Punkt 1 wieder begonnen.

Das geht nach diesem Ablauf Slot für Slot, Zyklus für Zyklus weiter.

Dieses Verfahren hat relativ hohe Kosten für den Suchvorgang, was an der Bild 6 auch zu erkennen ist. Hier gibt es hohe Speichernutzungsspitzen für den Buffer, die aber ab dem Datentransfer (Playback-Point) jeweils abnehmen. Im Ganzen ist der Speicherbedarf bei dieser Methode trotzdem effizient und klein gehalten.
Die Nachteile dieses Verfahrens sind, dass es keine IO-Zeit-Regulierbarkeit besitzt (Fixed) und die Disk-Latency, also die Zugriffszeit sehr hoch ist (Stretch). Demgegenüber stehen die

Vorteile, vor allem der geringe Speicherbedarf und das Speicher-Sharing, denn hier wird freigewordener Speicher als Bufferspace für den nächsten Slot vorgesehen, so dass mehrere Slots sich den Speicher im Parallelbetrieb teilen. Am besten erkennt man das an einer Grafik (Bild 6) .

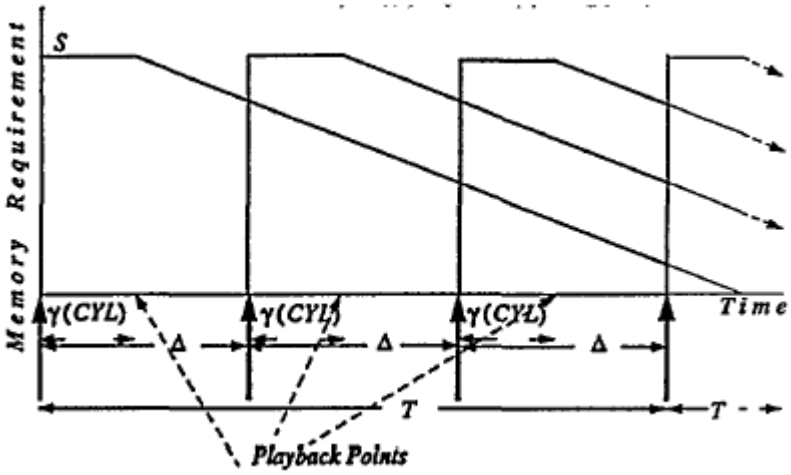

Bild 6: Service-Slots beim Fixed-Stretch. Quelle: [8]

4.3.3. BubbleUp

BubbleUp [8] verwendet das Fixed-Stretch-Verfahren, verkürzt aber noch weiter die Initial-zugriffszeiten, also der Zeitraum zwischen dem ankommenden Stream-Request und der ersten Auslieferung von Daten für den Stream. Die Verarbeitung von Slots bzw der Umgang mit freien Slots wird mit BubbleUp noch um einiges optimiert.

Hierzu ein Beispiel „Büroalltag“ zur Verdeutlichung:

Eine Woche wird in Arbeitstage unterteilt (Zyklen), die wiederum jeweils in Stundenblöcke unterteilt sind (Slots). Jetzt wird um 9 Uhr ein wichtiger Besucher (Stream-Request) erwartet, der aber um Punkt 9 Uhr nicht gekommen ist. Um 10 Uhr ist eigentlich ein Block Arbeiten geplant.

Jetzt könnte man ja den Block Arbeiten von 10 Uhr auf 9 Uhr legen, und so um 10 Uhr für den wichtigen Besuch Zeit zu haben. Ist er um Punkt 10 Uhr noch nicht da, kann der Arbeitsblock um 11 Uhr vorgelegt werden u.s.w. Hier „Bubble-Up“ der freie Block immer weiter zu einem späteren Zeitpunkt.

Somit könnte gewährleistet werden, dass jederzeit gearbeitet wird, um schneller mit den Arbeiten fertig zu sein.

Das BubbleUp-Verfahren ist allerdings um

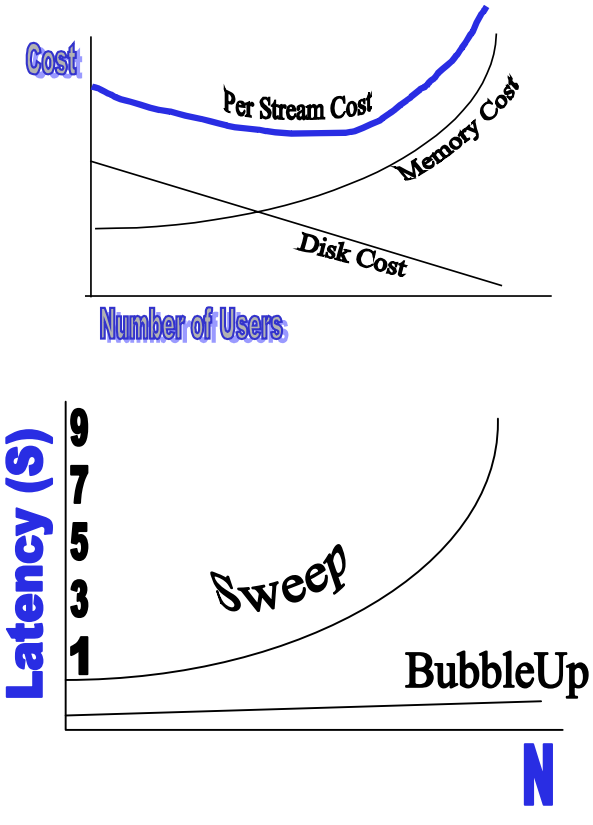

Bild 7: Kosten und Latency bei BubbleUp Quelle: [17]

einiges komplexer, weil die IO-Zeiten der Hardware noch mitbeachtet werden müssen u.s.w. Bei dem Bubble-Up-Verfahren wird auch ein Slot nach dem anderen bearbeitet. Ist zu Beginn eines leeren Slots kein Request angekommen (und es existiert beispielsweise bisher nur ein einziger Request), wird der letzte Request auf diesen aktuellen Slot verschoben, und wird somit weiterverarbeitet. Tritt zum Anfang des freien Slots ein neuer Request ein, wird dieser in diesem Slot bearbeitet, der ältere Request bleibt auf dessen Slot geparkt. Ist zu Anfang kein neuer Request gekommen und der Slot ist frei, aber im nächsten Slot ist noch ein Request geparkt, wird dieser vorgeschoben in die aktuelle, und der freie Slot verschiebt sich damit um eins nach hinten („BubbleUp“). Das geht immer so weiter. Eine kleine Ausnahme bilden Fast-Scan-Requests: diese werden mit höherer Priorität verarbeitet, weil sie mehr Ressourcen in Anspruch nehmen. Hier wird, wenn beispielsweise alle Slots mit Stream-Requests belegt sind und einer der Requests ein Fast-Scan-Request ist, diese auch im nächsten Slot, wenn eigentlich ein „normaler“ Request dran wäre, mit dem Fast-Scan-Request belegt, damit dieser zuerst bearbeitet wird. Allerdings wird hier für die anderen „normalen“ Requests eine Slot-Einheit Zeit damit weggenommen, so dass es für das Versenden der Daten der Streams hier zu einer Datenlücke kommt. Bubble-Up verhindert dies, indem die Segmentgröße verändert wird, und ein weiterer Slot hinzugefügt wird, so dass auch neue Requests (bei Bedarf) bearbeitet werden können.

4.4. Smoothing of Variable-Bit-Rate (VBR) Streams

4.4.1. Server-Push-Model

Ein Server nimmt ein Request eines Clients an, und zum Abspielen des Streams schickt der Server periodisch Daten zu dem Client. Das passiert so lange, bis das Ende des Streams erreicht ist, oder der Client die Wiedergabe abbricht. Der Datentransfer findet in Runden mit fester Dauer statt. Dabei werden die Daten von den Festplatten in Memory-Buffer gelesen, der für jeden Client reserviert wird, und anschließend über das Netzwerkinterface versendet. Die Größe der Daten, die in einer Runde gesendet werden, hängt zum Einen von der Decoding-Frame-Rate des Streams und von der Ressourceneinstellung des Servers ab. Gängigerweise wird eine Größe genommen, dass im ersten Schritt die Daten gelesen werden, im Zweiten versendet, und im Dritten vom Client dekodiert werden kann. Das Stream-Format ist meist MPEG-2, es gehen aber auch andere Formate, die konstante oder variable Bit-Rates haben. Requests werden hier von einem Admission-Control-Modul an wenig-ausgelastete Server weitergeleitet, die die Wiedergabe des Streams garantieren können. Eine Schedule-Datenbank speichert Informationen darüber, wie viel Daten in einer Runde von jeder Festplatte gelesen werden müssen und wie viel an die Clients versendet werden kann. Diese Informationen wird beim ersten Anlegen eines Stream-Objektes angelegt. Es ist üblich, dass Kopien der Stream-Objekte mit verschieden korrespondierenden Schedules existieren. [4]

4.4.2. Stride-Based-Allocation

Der Festplattenspeicher ist hier in großen Chunks mit festgelegter Größe unterteilt, sie werden hier Strides [4] genannt. Die Größe der Strides muss größer gewählt werden als die größte

Stream-Request-Größe in einer Runde sein kann. Diese Größe ist bekannt, da gespeicherte Streams der Reihe nach mit vorbestimmter Rate aufgespielt werden.
Wenn ein Stream-Request eintrifft, muss während einer Runde nur die empfangene Menge an Daten in den Speicher geholt werden, und nicht der gesamte Stride.
Da die interne Fragmentation wegen der (größeren) Größe des Streams (relativ gesehen zu den Strides) unwesentlich ist und außerdem ein Stride Daten für mehr als eine Runde vorsieht, kann es zu keiner externen Fragmentation bei der Stride-basierten Einteilung kommen.

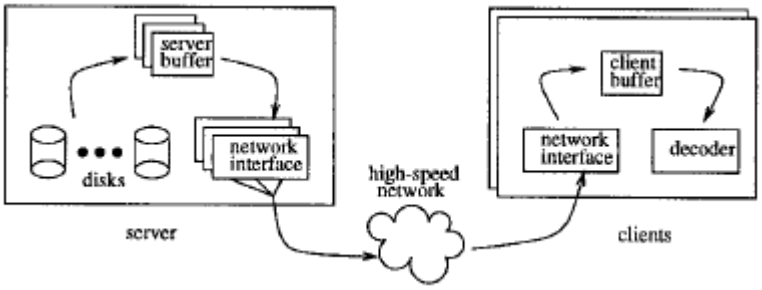

Bild 8: Server und Client beim Stream-Empfang. Quelle: [4]

Komprimierte Videostreams sind auf einem Multimedia-Server auf vielen Festplatten verteilt. Es können sehr viele Clients mittels einem High-Speed-Netzwerk auf alle diese Objekte zugreifen.

4.5. Beispiel Rechenzentrum Mannheim

Ein praktisches Beispiel ist das Rechenzentrum der Universität Mannheim. Dort wird ein Server verwendet, der an ein externes Storage-System (im Tera-Byte-Bereich) angeschlossen ist. Hier läuft auch ein Load-Balancing-Modul im Hintergrund. Laut Hersteller sind mit diesem Gerät weit über 1000 28.8kbit-Streams gleichzeitig möglich. Dieser Server soll zukünftig an die dotLrn-Plattform des Rechenzentrums, welche zum Speichern und Abrufen von Vorlesungsunterlagen eingesetzt wird, angeschlossen werden, so dass die Dozenten den Studenten zukünftig Ihre Vorlesungen auch als Videostream mit anbieten können. So können beispielsweise mehrere Studenten auch problemlos gleichzeitig auf einen einzigen Videostream einer Vorlesungen über das Internet zugreifen. Diese Anwendungsmöglichkeit wird sicherlich in der Zukunft weiter ausgebaut werden.

5. Zusammenfassung

Es gibt sowohl im Webseiten-Betrieb als auch im Multimedia-Betrieb eines Content-Servers mehrere Möglichkeiten, die Verfügbarkeit, Zuverlässigkeit, Geschwindigkeit und Skalierbarkeit zu optimieren. Eine „optimale“ Lösung gibt es so nicht, jede hat seine Vor- und Nachteile. Man kann aber, wenn mehrere gute Lösungen zusammen arbeiten können, schon einiges erreichen. In diesem Gebiet wird gerade heute durch den starken Zuwachs an Multimedia-Anwendungen im Internet jetzt vermehrt geforscht und neue Lösungen entwickelt, wobei es

jetzt immer zu ähnlichen wie den hier vorgestellten Lösungen hinauslaufen wird, da die physikalischen Gegegebenheiten des Internets zwischen Server und Client keine großen Sprünge ermöglichen.

Hier wurde ein kleiner Überblick an Möglichkeiten aufgezeigt, an welchen Stellen ein Content-Server optimiert werden kann - sowohl für dynamische Webseiten, als auch für multimediale Inhalte wie für das Videostreaming. Schwerpunkte liegen hier bei der infrastrukturellen Einrichtung, der Verwendung von Load-Balancing und Admission-Control-Modulen, lokalen Prioritätszuordnungen und der Vorstellung mehrerer Streamingverfahren.

Die heutige Gesellschaft steigt mehr und mehr in das Internet ein, und erwartet nun eine möglichst hundertprozentige Verfügbarkeit aller Dienste, was ohne das Verwenden von speziellen Algorithmen und System beispielsweise zur Lastverteilung bei Servern nicht mehr möglich ist.

6. Literaturnachweis

[1] Z. Ge, P. Ji, P. Shenoy. *A Demand Adaptive and Locality Aware (DALA) Streaming Media Server Cluster Architecture.*
[2] Y. Ruan, V. S. Pai. *The Origins of Network Server Latency & the Myth of Connection Scheduling*
[3] S. Elnikety, E. Nahum, J. Tracey, W. Zwaenepoel. *A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites*
[4] S. Anastasiadis, K. C. Sevcik, M. Stumm. *Server-Based Smoothing of Variable Bit-Rate Streams.*
[5] R. Pandey, J.F. Barnes, R. Olsson. *Supporting Quality Of Service in HTTP Servers*
[6] J. Santos, R. R. Muntz, B. Ribeiro-Neto. *Comparing Random Data Allocation and Data Striping in Multimedia Servers.*
[7] L. Amini, A. Shaikh, H. Schulzrinne. *Modeling Redirection in Geographically Diverse Server Sets*
[8] E, Chang, H. Garcia-Molina. *BubbleUp: Low Latency Fast-Scan for Media Servers*
[9] S. Berson, L. Golubchik, R. R. Muntz. *Fault Tolerant Design of Multimedia Servers*
[10] V. Cardellini, E. Casalicchio, M. Colajanni, P. S. Yu. *The State of the Art in Locally Distributed Web-Server Systems*
[12] J. Aweya, M. Oullette, D. Y. Montuno, B. Doray, K. Felske. *An adaptive load Balancing scheme for web servers.*
[13] D. Le Moal, T. Takeuchi, T. Bandoh. *Cost-Effective Streaming Server Implementation Using Hi-Tactix.*
[14] L. Cherkasova. *Scheduling strategy to improve response time for Web applications*. In *Proceedings High Performance Computing and Networking (HPCN)*, Amsterdam, April 1998
[15] D.-Y. Ahn, Kyu-Ho Park. *Disk Allocation Methods Using Genetic Algorithm*
[16] J. R. Santos, R. Muntz. *Design of the RIO (Randomized I/O) Storage Server*
[17] E. Chang, H. Garcia-Molina. *Effective Memory Use in a Media Server*
[18] J. Rexford, D. Towsley. *Smoothing Variable-Bit-Rate Video in an Internetwork*
[19] S. Sen, D. Towsley, Z.-L. Zhang, J.K. Dey. *Optimal Multicast Smoothing of Streaming Video over an Internetwork*
[20] E. Chang, H. Garcia-Molina. *On Managing Continuous Media Data.*