Web Server Architektur

"Cooperative Caching
in a Cluster Environment"

Seminararbeit
Von
Daniel Förderer
aus
Mannheim

vorgelegt am
Lehrstuhl für Praktische Informatik IV
Prof. Dr. W. Effelsberg
Fakultät für Mathematik und Informatik
Universität Mannheim

Juli 2004

Betreuer: Dipl.-Wirtsch.-Inf. Jürgen Vogel

1 Einleitung

Web Services sind modulare Anwendungen, die einer exakten und fest vorgelegten Beschreibung genügen. Sie werden über ein Netzwerk, im Normalfall das World Wide Web [1] publiziert, lokalisiert und ausgelöst.

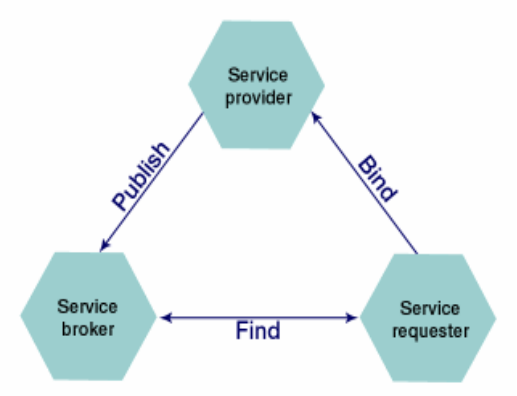

Abbildung 1 - Die 3 Rollen eines Web Services

Die Web Service Architektur beschreibt drei Rollen (Abb.1): Den “Service Provider“, den “Service Requester“ und den “Service Broker“. Diese Rollen sind über drei Basisoperationen miteinander verbunden: “publish“, “find“ and “bind“. Eine Netzwerkkomponente kann jede dieser Rollen spielen.
Es gibt zwei separate Dokumente, die den Web Service vollständig beschreiben:

- “Well-Defined Service” (WDS)
- “Network-Accessible Service Specification Language” (NASSL)

Das WDS Dokument enthält nonoperable Service Informationen, wie Kategorie, Service Beschreibung und Auslaufdatum, aber auch Informationen über den Service Provider, wie dessen Firmenname, Adresse und Kontaktdaten.
Das NASSL Dokument beschreibt die genaue Funktionalität des Services, wie zum Beispiel das Service Interface, Implementierungsdetails und Zugangsprotokolle.
Web Services können dynamisch in Applikationen eingebunden werden indem ein kapazitätsgestützter “Look-up“ anstatt einem traditionellen statischen Binden während der Laufzeit durchgeführt wird. Die dynamische Natur der Zusammenhänge erlaubt es den Implementierungen plattform-, programmiersprachen- und kommunikationsmechanismusunabhängig zu sein.
Damit all diese Kommunikation jedoch fehlerfrei ablaufen kann, ist es wichtig alle Schnittstellen genau zu definieren. Dies geschieht mit der “Web Service Description Language“ [2] (WSDL).

Damit es zu einer Nutzung eines Web Services kommt, sind mehrere Schritte notwendig (Abb.2).

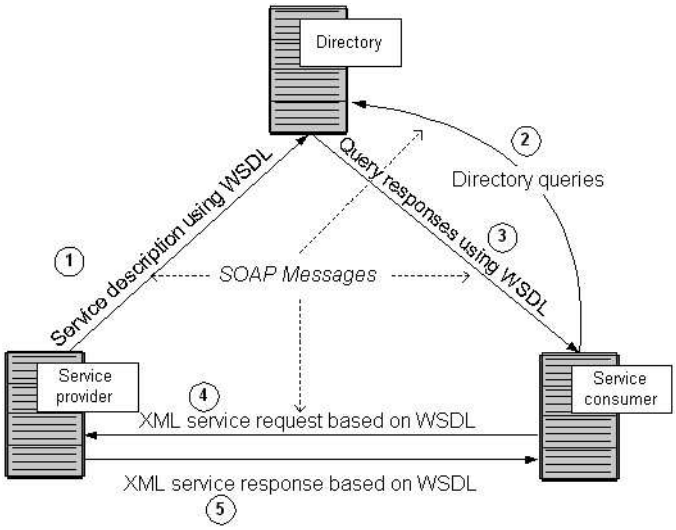

Abbildung 2 – Ablauf eines Web Services

1. Der Service Provider beschreibt seinen Service mit Hilfe von WSDL. Diese Definition wird dann in einem allgemeinen Register, zu vergleichen mit den gelben Seiten, abgelegt. “Universal Description, Discovery and Integration“ [3] (UDDI) ist ein solches Register.
2. Der Service Consumer sendet eine Anfrage an das Register um einen passenden Service zu lokalisieren und festzustellen, wie er mit diesem kommunizieren kann.
3. Ein Teil der vom Service Provider bereitgestellten WSDL wird nun zum Service Consumer geschickt. Somit kann der Service Consumer erkennen, wie er die Anfragen an den Service Provider formulieren muss und in welchem Format dieser dann die Antworten sendet.
4. Der Service Consumer benutzt seinerseits nun WSDL um eine korrekt formulierte Anfrage an den Service Provider zu schicken.
5. Der Service Provider bearbeitet die Anfrage und sendet dem Service Consumer die Antwort.

Alle für im Verlauf gesendeten Informationen, werden mit dem “Simple Object Access Protocoll“ [4] (SOAP) gesendet. SOAP benutzt das “Hyper Text Transport Protocoll“ [5] (HTTP) und dient als Umschlag für die Web Service Nachrichten, welche in der “Extensible Markup Language“ [6] (XML) formuliert sind. Die genaue Syntax dieser Nachrichten ist mittels WSDL genau definiert und wird von SOAP dann nochmals gekapselt.
Es besteht ebenso die Möglichkeit mehrer Services zu bündeln und den daraus resultierenden Web Service noch mächtiger zu machen. Man spricht in diesem Zusammenhang von einer Komposition und bedient sich dabei der “Business Process Execution Language for Web Services“ [7] (BPEL4WS).
Für die Bearbeitung eines Web Service, also der Anfragenauswertung, Informationsbeschaffung und Anfragenbeantwortung, ist neben dem Code

auch eine Maschine notwendig, die dies erledigt. Eine Solche Maschine nennt man Web Server.

2 Web Server

Ein Web Server ist ein Rechner in einem Netzwerk, der Webseiten, bzw. Daten für Web Clients bereitstellt. Clients können diese Daten beispielsweise über normale HTTP Anfragen oder über Web Services erfragen.
Im Allgemeinen gibt es zwei Verständnisse für den Begriff Web Server. Zum einen wird hiermit oft der Rechner bzw. die Maschine selbst bezeichnet, auf der die entsprechende Software läuft. Zum anderen wird die befähigende Software selbst Web Server genannt (zum Beispiel der Apache Web Server [8]).
Der Web Server muss auf Client Anfragen reagieren und diese bearbeiten.

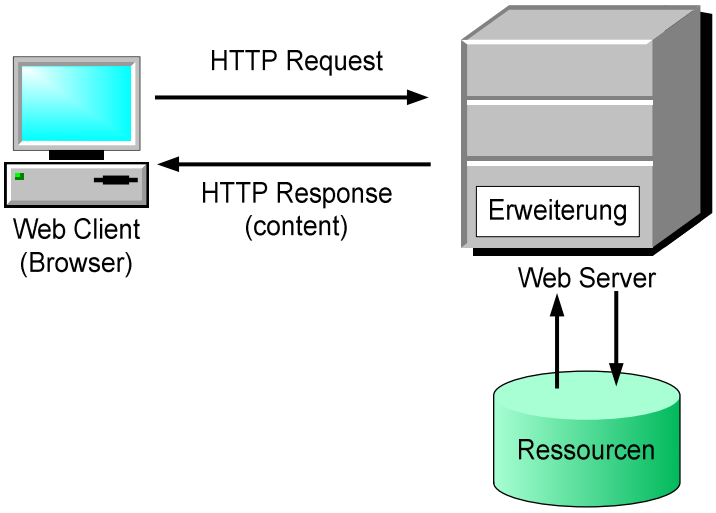

Abbildung 3 - Ablauf einer HTTP Anfrage an einen Web Server

Sobald ein Client dem Web Server eine HTTP Anfrage schicken will, bauen beide eine “Transport Control Protocoll“ [9] (TCP) Verbindung auf (Abb.3). Über diese TCP Verbindung schickt der Client dann seine HTTP Nachricht. Diese Anfrage kann eine SOAP Nachricht beinhalten, die das WSDL eines Web Services verlangt. Der Server löst daraufhin seinerseits die der Anfrage zugeordneten Prozesse aus und schickt nach erfolgreicher Bearbeitung die Ergebnisse zurück. Sollte die Bearbeitung der Anfrage einen Fehler verursachen, oder gar nicht bearbeitbar sein, wird dem Client eine Fehlermeldung mit speziellem Errorcode geschickt [10]. Ist alles ordnungsgemäß verlaufen, kommt es am Ende zu einem beidseitigen Verbindungsabbau. Dies ist allerdings nicht zwingend.
Die Performance eines Web Servers kann durch Erweiterungen noch gesteigert werden, beziehungsweise kann sein Verhalten dynamisch bestimmt werden. Diese Erweiterungen ermöglichen einen noch effizienteren bzw. gezielten Zugriff auf die Daten aus der Datenbank. Eine solche Erweiterung ist ebenfalls nötig, wenn es um die dynamische Darstellung von Web Objekten geht. Beispielsweise enthält die eine Webseite, welche aktuelle Sportergebnisse darstellt, sowohl statische, als auch dynamische Web Objekte. Ein statisches Objekt wäre in diesem Beispiel wohl das Logo des Betreibers und die aktuelle Ergebnisliste wäre ein dynamisches Objekt. Für den Client spielt es jedoch keine Rolle, welcher Art an Objekten der Server nun in einer Webseite verbinden muss.

Eine Möglichkeit ist, die ankommenden HTTP Anfrage von einem Java Programm, einem so genannten Servlet mit speziellen Methoden, beantworten zu lassen. So kann eine dynamische und individuelle Antwort für den Client generiert werden.
Weiter besteht allerdings auch die Möglichkeit, zur Laufzeit ausführbaren Java Code in statische HTML-Seiten einzubetten. Seiten, die dynamischen Java Code enthalten, werden als “JavaServer Pages“ [11] (JSP) bezeichnet. Beim Aufruf einer JSP wird daraus ein Servlet generiert (compiliert), welches dann durch die Servlet Engine ausgeführt wird.
Eine weitere Möglichkeit ist es, ein in HTML eingebettetes PHP [12] Programm beim Aufruf zu interpretieren um so dynamische Zugriffe auf die Ressourcen im Backend zu ermöglichen. Weitere Möglichkeiten wären noch CGI [13] Skripte und “Active Server Pages“ [14] (ASP).

3 Cluster

Da ein Web Server mit einer immensen Anfragenlast in kurzer Zeit konfrontiert wird, muss er mit ausreichenden Ressourcen, sprich Rechenleistung und Speicher, ausgestattet sein. Um nun viele Anfragen gleichzeitig zu bearbeiten, besteht die Möglichkeit die zur Verfügung stehende Rechenleistung durch den Zusammenschluss mehrerer Server in einem "Cluster" zu Erhöhen.
Unter einem Cluster versteht man formal eine lose gekoppelte Gruppe von Servern, die auf dieselben Daten zugreifen und dieselbe Gruppe von Clients bedienen können.
Bei diesem Zusammenschluss werden mehrere unabhängige Web Server hinter einem Lastverteiler, dem "Dispatcher", an das Netz angeschlossen. Dieser Dispatcher ist verantwortlich für die Verteilung der ankommenden Anfragen. Er ist zu jedem Zeitpunkt über den Auslastungsgrad der einzelnen Server informiert. Somit kann eine ankommende Anfrage immer an den Server weitergeleitet werden, der gerade die kleinste Auslastung hat. Dadurch lassen sich einzelne Überlastungen und die daraus resultierenden langen Wartezeiten verhindern.
Da Web Objekte nach Möglichkeit zu jeder Zeit verfügbar sein sollen, darf es in keinem Fall zu einem Ausfall des von einem Web Server zur Verfügung gestellten Service kommen. Diesem Risiko kann in einem Cluster äußerst effizient begegnet werden. Fällt dort eine einzelne Maschine aus, kann bis zu deren Reparatur die Last auf die verbleibenden Maschinen verteilt werden. Genauso können Wartungstätigkeiten ohne Probleme durchgeführt werden.
Der Zusammenschluss von Web Servern bietet somit neben einer Leistungssteigerung auch ein höheres Maß an Zuverlässigkeit gegenüber einem einzelnen Server.

Die Möglichkeiten, die eine solche Architektur bietet, kann im Detail in der Arbeit von Frau Aysen Shibla "Web Server Architektur – Clustering" nachgelesen werden.

4 Caching

Als Cache bezeichnet man den Speicher eines Rechners, der schnell adressierbar ist und für das Zwischenspeichern der zuletzt bzw. der am Öftesten aufgerufenen Daten verwendet wird. Dieser Speicher existiert auch auf einem Server, wird dort allerdings für die Bearbeitung von ankommenden Anfragen verwendet.

4.1 Allgemein

Die Performance eines Web Servers wird im Normalfall nicht an dessen Rechenleistung gemessen, sondern in der Zeit, die er im Schnitt benötigt, um Clientanfragen zu beantworten. Um diese Antwortzeit zu verringern, empfiehlt es sich eine Kopie der oft benötigte Daten bzw. Webseiten in einem schnellen Speicher prozessornah zwischenzuspeichern. Auf diesen Speicher kann schnell und ohne Netzwerklast zugegriffen werden. Dieser Speicher wird als Cache bezeichnet. Das Verfahren selbst wird “Caching“ genannt.

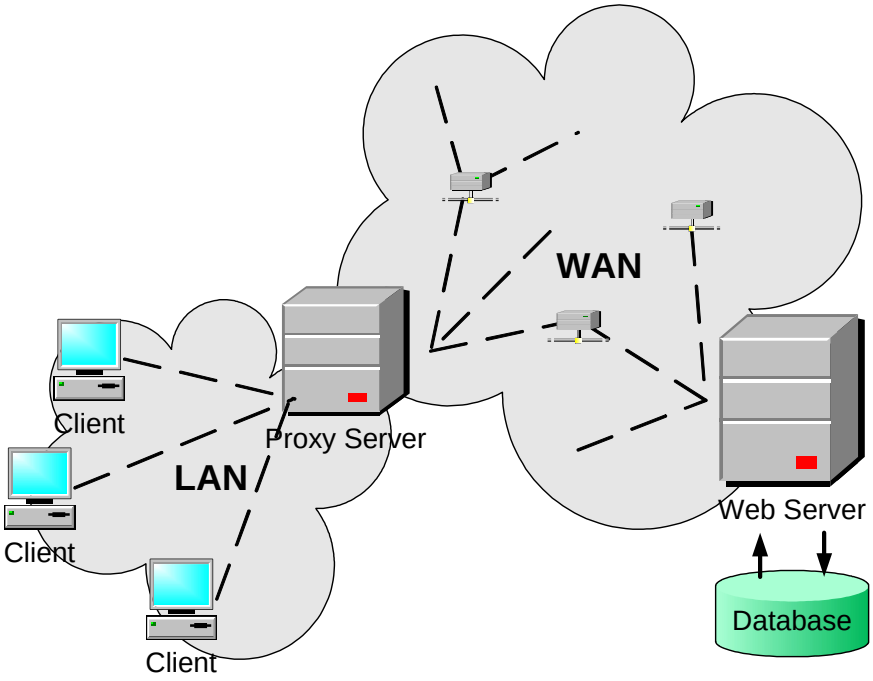

Abbildung 4 - "Caching"-Möglichkeiten

Die erwähnte Antwortzeit kann man auf drei Arten verringern (Abb.4):

- Daten bereits auf der Clientmaschine im Browser Zwischenspeichern
- Daten auf einem, zwischen Client und Web Server liegenden, Proxy Servers zwischenspeichern
- Daten auf dem Web Server zwischenspeichern

Die Antwortzeit verringert sich aus dem Grund, da die Daten nicht erst wieder aus der räumlich entfernten Datenbank geladen werden müssen. Mögliche Anfragen können so beispielsweise bereits auf der Clientmaschine bearbeitet werden, da das gesuchte Web Objekt bereits schon einmal geladen wurde und sich noch im Speicher befindet. Nebenbei kann man durch das

Zwischenspeichern von Web Objekten im Client Browser oder auf dem Proxy Server die Netzlast senken.
Dieses Paper beschäftigt sich allerdings ausschließlich mit dem Zwischenspeichern oft benötigter Daten im Cache eines Web Servers bzw. mehrerer Web Server in einem Cluster.

Der Caching Algorithmus eines Web Servers muss seinen Inhalt möglichst so verwalten, sodass die Anzahl der Fernzugriffe auf die Datenbank minimiert werden.
Web Objekte sind die kleinsten und nicht weiter teilbaren Einheiten einer Webseite bzw. eines Datensatzes. Sie besitzen eine völlig andere Granularität als herkömmliche Files. Während herkömmliche Files in Blöcke fester Größe zerlegt werden können, ist dies mit Web Objekten nicht möglich. Diese haben eine feste Größe und sind entweder vollständig oder gar nicht im Cache enthalten.
Ein Algorithmus, welcher den Inhalt eines Caches kontrolliert, muss abwägen, ob er viele kleine Web Objekte oder eher wenige große Objekte im Cache halten will. Viele kleine Objekte im Cache würden eine höhere „Hit-Rate“ und somit weniger Fernzugriffe auf die Datenbank bedeuten. Von einem Hit spricht man, wenn eine Anfrage ein Objekt anfordert, dass sich noch im Speicher befindet. Andererseits dauert es wesentlich länger ein großes Objekt von der Datenbank zu laden, als ein kleines.
Neben dieser Wahl muss ein Speicherkontrollalgorithmus allerdings noch weitere Entscheidungen bezüglich seiner Zusammensetzung treffen. Web Objekte können in zwei Gruppen untereilt werden. Es gibt statische und dynamische Web Objekte. Statische Objekte besitzen keine maximale Lebensdauer. Sie ändern ihren Zustand über den Zeitverlauf nicht. Diese Objekte werden also lediglich aus dem Cache geworfen, wenn ihre Anfragenanzahl über einen bestimmten Zeitraum niedriger ist, als die aller anderen Objekte im Cache. Dynamische Objekte hingegen können ihren Zustand über den Zeitverlauf ändern. Deshalb wird ihnen eine maximale Lebenszeit mit angehängt. Spätestens nach dieser Zeit muss der Web Server prüfen, ob die Version, die er in seinem Speicher hält noch aktuell ist. Diesen Vorgang nennt man Validation. Die Spezifikation von HTTP/1.1 [10] hält hierfür ein einfaches Modell bereit, welches mittels eines Ablaufdatums den Web Servern hilft, ihren Cache aktuell zu halten.

4.2 Cooperative Caching

Da die Rechenleistung eines Servers dennoch nicht unerheblich für die Antwortzeit ist, werden Servermaschinen oft in einem Cluster gebündelt und organisiert.
Dieses Clustering hat zur Folge, dass man nun deutlich mehr Prozessorleistung für die Behandlung der Clientanfragen zur Verfügung hat. Mehrere tausend Anfragen in der Sekunde wären für einen einzelnen Web Server deutlich zu viel. Die Antwortzeit würde sich dann aufgrund des Wartens auf CPU-Zeit drastisch erhöhen. Aus diesem Grund werden mehrere Web Server im Cluster parallel geschaltet und die Anfragenlast mit einem vorgelagerten Dispatcher verteilt. Es verbessert das Resultat allerdings nur bedingt, wenn man die Servermaschinen lediglich mit einer besseren Lastverteilung im Cluster hinter einen Dispatcher hängt. Das würde einzig und

allein die Zeit reduzieren, die eine mögliche Überlastung eines Servers zur Folge hätte. Die Zeit für eine einzelne Anfrage würde so in keinem Fall reduziert werden. Aus diesem Grund bietet es sich an einen Caching Algorithmus zu entwickeln, der die Speicher der Server im Cluster intelligent verwaltet. Ein solcher Algorithmus ist Kern des “Cooperative Caching“ [15].
Die Grundidee des Cooperative Caching liegt darin, das in einem Cluster der Cache eines jeden Web Servers für alle anderen Server im Cluster sichtbar und zugänglich ist. Dieser, durch Zusammenlegung entstandene, virtuelle Cache wird als “Global Object Space“ (GOS) bezeichnet.
Alle im Cluster hängenden Web Server stellen einen Speicherblock bestimmter Größe für den gemeinsamen Cache des Clusters zur Verfügung. Der GOS beinhaltet somit alle Web Objekte die in den einzelnen Caches aller Web Server präsent sind.
Der eigene Cache eines jeden im Cluster hängenden Web Servers wird als “Hot Object Space“ (HOS) bezeichnet. Er ist für alle im Cluster hängenden Web Server sowohl sichtbar, als auch zugänglich. Die Sichtbarkeit und der Zugang werden über einen speziell entworfenen “Lookup-Mechanismus“ geregelt. Alle HOS zusammen bilden somit den GOS.
Um den Fall auszuschließen, dass mehrere Server im Cluster ein und dasselbe Web Objekt aus dem Backend laden, werden alle auf dem Backend existierenden Objekte über eine Partitionsliste eindeutig einem “Home Node“, also einem Web Server, im Cluster zugeordnet. Dieser Server besitzt als einziger das Recht einen Fernzugriff auf das Objekt durchzuführen. Neben diesem Zugriffsrecht muss dieser Server alle globalen, mit dem Web Objekt zusammenhängenden Parameter und Werte verwalten. Der Home Node eines Objekts weiß zu jedem Zeitpunkt, in welchem HOS das Web Objekt präsent ist. Aus diesem Grund ist er für die globale Zugriffsstatistik des Objekts verantwortlich.

Jedes Web Objekt besitzt zwei Zähler:

- “Local Access Counter“ (LAC)
- “Approximated Global Access Counter” (AGAC)

Jeder HOS zählt lokal die Zugriffe auf jedes in ihm enthaltene Web Objekt. Dieser Wert dient dazu den AGAC des Objekts zu errechnen. Der AGAC ist ein Zugriffszähler für jedes Web Objekt über den ganzen Cluster. Er wird vom Home Node des Objekts verwaltet und berechnet. Hierfür muss der LAC zum Home Node gesendet werden. Dieses Senden erfolgt entweder periodisch nach einem vom Administrator festgelegten Zeitintervall oder sobald der LAC einen ebenso festgelegten Schwellenwert überschritten hat. Das ist nötig, um den AGAC so aktuell wie möglich zu halten ohne das Netzwerk durch kontinuierliche Updatemeldungen unnötig zu belasten. Der Home Node sendet den neuen AGAC nach einem Update wieder an den HOS zurück. Der HOS ersetzt den alten AGAC des Objekts mit dem neu erhaltenen und fängt wieder von vorne an den LAC hoch zu zählen. Der AGAC akkumuliert somit alle Zugriffe auf das Web Objekt in den einzelnen HOS’s und ist somit die Summe aller LAC’s des Objekts.
Nun wird klar, dass der AGAC der entscheidende Faktor für das Management der im Cache enthaltenen Objekte ist. Die Objekte in einem HOS werden in einer Liste organisiert. Je niedriger der AGAC eines Objekts ist, desto weiter

unten in der Liste steht es. Das letzte Web Objekt in der Liste muss dann zugunsten eines neuen Web Objektes weichen. Ein neues Web Objekt wird dann aus der Datenbank geladen, wenn die Bearbeitung einer Anfrage diesem Web Objekt bedarf und es sich nicht im Cache eines Web Servers im Cluster befindet.

Bis jetzt wurde allerdings nur der Zugriff auf die Web Objekte in der Datenbank organisiert, aber noch nicht wirkliche Zeit eingespart. Ganz im Gegenteil: Es erscheint so, als ob bereits der Dispatcher wissen müsste, welcher Web Server Home Node für das angeforderte Web Objekt ist. Oder was passiert gar, wenn eine Anfrage für zwei Objekte von unterschiedlichen Home Nodes beantwortet werden muss? Wie bereits eingangs dieses Kapitels erwähnt ist der HOS eines jeden Servers im Cluster für die anderen Server sichtbar und zugänglich. Kommt nun eine Anfrage für ein Web Objekt an, das der bearbeitende Server nicht in seinem Cache hat, kann er prüfen, ob ein anderer Server es in seinem Cache hält und es von dort aus laden. Für den Prüfvorgang verschickt der suchende Server kurze Nachrichten an alle anderen Server im Cluster. Hat ein anderer Server das Objekt bereits im Cache, so sendet er dieses an den suchenden Server, welcher immer noch bearbeitender Web Server für die Clientanfrage ist. In diesem Fall ist das Web Objekt nun in zwei Caches enthalten. Dieser Vorgang ist immer noch wesentlich schneller, als das Laden des Objekts von einer räumlich entfernten Datenbank.

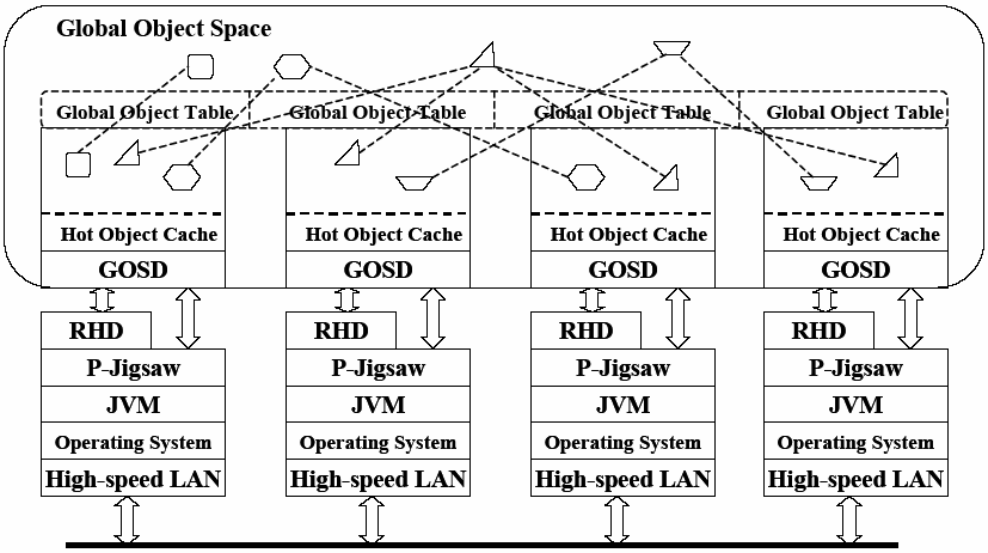

Abbildung 5 - Global Object Space (GOS) Modell

Jedoch muss auch dieser verteilte Caching Algorithmus organisiert sein. Aus diesem Grund operiert jeder Web Server im Cluster zwei Server Prozesse (Abb.5), so genannte “Deamons“. Zum einen den “Global Object Space Deamon“ (GOSD) und zum anderen den “Request Handling Deamon“ (RHD). Diese beiden Prozesse stellen die Arbeitsweise des Web Servers sowohl innerhalb des Clusters, als auch nach außen hin sicher.
Der GOSD ist hauptsächlich für den HOS des jeweiligen Web Servers verantwortlich. Er sichert dessen Transparenz und Zugriffsmöglichkeit für alle anderen Web Server des Clusters. Der GOSD bearbeitet sowohl die

ankommenden Anfragen vom Web über den RHD, als auch die der anderen Web Server innerhalb des Clusters
Gleichzeitig verwaltet der GOSD zwei Tabellen welche wichtig für das lokale Cachemanagement sind:

- “Local Object Table” (LOT)
- “Global Object Table” (GOT)

Auf dem LOT sind alle Informationen der im HOS präsenten Web Objekte gespeichert. Hierzu gehören deren AGAC, LAC und die ID des jeweiligen Home Node. Diese ID ist wichtig um dem Home Node den aktuellen LAC zukommen zu lassen.
Der GOT enthält alle Informationen, welche der Server als Home Node über seine, ihm zugeordneten, Web Objekte wissen muss. Dazu gehören sämtliche ID’s der anderen Web Server im Cluster, die AGAC der Objekte für den der Server Home Node ist und die ID’s der Server in deren Cache die Objekte gerade präsent sind.
Für jegliche Kommunikation mit dem Web ist der RHD zuständig. Er bearbeitet alle über den TCP Port ankommenden HTTP Anfragen und leitet diese an den GOSD weiter.

4.3 Replacement Policies

Da nu die Organisation und die Verwaltung der Caches untereinander diskutiert wurde, soll nun erörtert werden warum ein Web Objekt aus einem Cache geworfen wird um ein neues aufzunehmen.
Die Aktualität eines Caches ist ebenso sicher zu stellen, wie dessen Zusammensetzung bezüglich der Anfragenanzahl auf die enthaltenen Objekte. Wie bereits angesprochen, muss zwischen statischen und dynamischen Web Objekten unterschieden werden. Es macht keinen Sinn dynamische Objekte nach Ablauf ihrer Lebenszeit ohne eine Überprüfung auf weitere Gültigkeit noch länger im Cache zu halten.
Der Zustand solcher Objekte muss demnach nach Ablauf der Lebenszeit geprüft werden. Ein Objekt, welches nicht mehr aktuell ist muss entweder erneuert, also auf der Datenbank validiert, oder aus dem Cache geworfen werden.
Generell lassen sich hierbei zwei Strategien unterscheiden:

- Passive Validation
- Aktive Validation

Bei der passiven Validation kommt es erst dann zur Validation eines Objekts, wenn eine neue Anfrage für das Objekt eingegangen ist. Mit anderen Worten gibt es bei dieser Methode keinen Deamon oder ähnliches, der alle Objekte im Cache kontinuierlich auf deren Validität bezüglich deren Lebenszeit prüft. Dieses Prinzip wird jedoch bei der aktiven Validation verfolgt. Hier werden die Objekte immer nach Ablauf ihrer Lebenszeit validiert, egal, ob gerade eine Clientanfrage dafür eingegangen ist oder nicht.
Eine Aktive Validation kann allerdings nicht unendlich oft erfolgen, da das ständige Erneuern des Web Objekts ohne Clientanfragen zu einem großen und möglicherweise unnötigem Mehraufwand führen könnte. Aus diesem Grund wurde ein Erneuerungszähler (“Renewal Credit“) eingeführt. Dieser

Zähler wird im Allgemeinen mit jedem aktiven Erneuern eines Web Objekts um eins herunter gezählt. Er fließt maßgeblich in die aktiven Cache Ersetzungsstrategien ein.
Sobald der Erneuerungszähler abgelaufen, also Null ist, wird das Objekt, wenn es nicht ohnehin bereits aufgrund einer niedrigen Anfragenanzahl aus dem Cache geworfen wurde, entfernt.
Allerdings wäre eine solche Verwaltung dieses Zählers in vielen Fällen ungenau und kann zu Zeiteinbußen führen. So könnten beispielsweise Web Objekte mit einer sehr langen Lebenszeit einen Vorteil gegenüber Objekten mit kurzer Lebenszeit haben, da diese wesentlich öfter validiert werden müssten. Somit könnte ein Web Objekt mit kürzerer Lebenszeit viel gefragter sein, jedoch aufgrund des sich schneller verringernden Erneuerungszählers aus dem Cache geworfen werden. Das Herunterzählen des Erneuerungszählers allein, ohne das Miteinbeziehen von Anfragenstatistiken, kann also in vielen Fällen zu einem falschen Tausch eines Objektes im Cache führen.
Aus diesem Grund gibt es für die genaue Verwaltung des Erneuerungszählers komplexere Algorithmen bzw. Verfeinerungen des trivialen Mechanismus. In diese fließen größtenteils auch die aktuellen Anfrageereignisse für das jeweilige Web Objekt mit ein.
Die möglichen Anfrageereignisse lassen sich in zwei Gruppen unterteilen. Die eine Gruppe dreht sich um den Inhalt (“Content“) des Caches. Die andere um die Validität (in diesem Fall “Freshness“) der darin enthaltenen Objekte. Insgesamt gibt es vier mögliche Ereignisse, die bei einer Anfrage eintreten können:

- Content Miss
- Content Hit
- Freshness Hit
- Freshness Miss

Ein “Content Miss“ bedeutet, dass das angefragte Objekt sich momentan nicht im Cache befindet und deshalb entweder von einem anderen Cache eines Web Servers oder direkt vom Backend geholt werden muss. Ein “Content Hit“ dagegen bedeutet, dass das Objekt aktuell im Cache vorhanden ist. Die beiden anderen Ereignisse, “Freshness Hit“ und “Freshness Miss“ können nur als weitere Ereignisse nach einem “Content Hit“ eintreten, da diese sich auf die Validität des im Cache vorhandenen Objekts beziehen. Ein “Freshness Miss“ bedeutet, dass das Objekt zwar im Cache vorhanden ist, seine Lebenszeit allerdings schon abgelaufen ist und das Web Objekt deshalb nun neu validiert werden muss. Ein “Freshness Hit“ führt zu einer sofortigen Beantwortung der Anfrage, da das Objekt erstens im Cache vorhanden und zweitens valide ist.
Insgesamt sollen hier vier weitere Algorithmen vorgestellt werden, die in der Praxis, je nach auf dem Web Server enthaltenem Datentyp, alle ihre Relevanz haben und Verwendung finden:

- **OPT(i)**
  Dieser, als utopisch anzusehender Algorithmus, geht davon aus alle zukünftigen Anfragen zu kennen. Somit wäre es möglich den Cache auf alle kommenden Anfragen vorzubereiten. Dieser Ansatz besitzt

aber dennoch eine praktische Relevanz, da man anhand ihm messen kann in welchem Maß ein anderer realisierbarer Algorithmus an dieses Optimum heranreicht beziehungsweise. wo noch Verbesserungen möglich sind.

- **RECENCY(*k*)**
  Bei diesem Ansatz, wird der Erneuerungszähler mit jeder neuen Anfrage wieder auf seinen Anfangswert k gesetzt. Dieses Rücksetzen ist somit gleichbedeutend mit dem Laden eines völlig neuen Objekts aus der Datenbank, welches ebenso mit dem maximalen Erneuerungszähler starten würde.

- **FREQ(*j,m*)**
  Dieser Algorithmus macht einen Unterschied, ob die ankommende Anfrage zu einem “Freshness Hit“ oder einem “Freshness Miss“ führt. Bei einem “Freshness Hit“ wird der Erneuerungszähler wieder auf seinen anfänglichen Wert m zurückgesetzt. Bei einem “Freshness Miss“ hingegen wird der Zähler lediglich wieder um j erhöht.

- **TH-FREQ(*th,m*)**
  Dieser Ansatz berücksichtigt das Verhältnis der Anzahl an Anfragen zu der noch verbleibenden Lebenszeit des Web Objekts. Fällt dieses unter eine vorgegebene Schranke (“Threshold“) th, wird der Erneuerungszähler heruntergezählt und das Objekt neu geladen. In anderen Worten führt dies dazu, dass Objekts mit einer niedrigeren Anfragenzahl schneller aus dem Cache verschwinden.

Man kann allerdings davon auszugehen, dass der Speicherplatz in den jeweiligen Caches sehr begrenzt im Vergleich zur absoluten Datenmenge auf der Datenbank ist und pro Sekunde eine sehr hohe Anzahl an verschiedenen Anfragen eingeht. Somit stellt sich in den meisten Fällen gar nicht die Frage den Inhalt des Cache aufgrund seiner Lebenszeit zu organisieren. Oftmals entscheidet die reine Zugriffsstatistik bereits darüber, welches Web Objekt im Cache nun einem neuen weichen muss. Es bestehen unter Einbezug der Anfragenstatistiken für die Web Objekte zwei simple Mechanismen:

- “Last Recently Used“ (LRU)
- “Least Frequently Used“ (LFU)

Der LRU Mechanismus organisiert die Liste eines Speichers so, dass immer das Web Objekt als nächstes verworfen wird, das am längsten schon nicht mehr nachgefragt wurde Unabhängig von seiner absoluten Anfragenzahl. Da der Speicher eines Web Servers jedoch nicht für die Bearbeitung einzelner Anfragen der unmittelbaren Vergangenheit den besten Inhalt haben soll, sondern für die Mehrheit in der Zukunft, wird schnell klar, dass dieser Mechanismus für den Caching Bereich nicht geeignet ist. Es könnte beispielsweise passieren, dass ein Objekt aus dem Speicher geworfen wird, welches eine sehr hohe absolute Nachfrageanzahl besitzt, jedoch gerade in der letzten Minute nicht angefragt worden ist. So könnte ein anderes Objekt im Cache bleiben, das lediglich ein einziges Mal, allerdings vor drei Sekunden, angefragt wurde.

Der LFU Mechanismus dagegen vergleicht und ordnet die im Speicher enthaltenen Objekte aufgrund der absoluten Nachfragezahl. Dieser Algorithmus ist auch nicht optimal, da so ein bereits sehr lange nicht mehr angefragtes Web Objekt länger im Speicher bleiben könnte, als ein neues, was vielleicht gerade in der letzten Zeit eine hohe Nachfrage erlebt hat. Aus diesem Grund gibt es von LFU noch zwei Erweiterungen:

- "LFU-Aging"
- "Weighted-LFU"

Die Idee, welche LFU-Aging zugrunde liegt ist, dass die Popularitäts-Informationen ohne Erneuerung über den Zeitverlauf immer unzuverlässiger werden. Es wird ein so genanntes "Aging the AGAC" durchgeführt. Der AGAC wird am Ende eines vom Administrator festgelegten Zeitintervalls halbiert.
Der Weighted-LFU Algorithmus dagegen zieht den unterschiedlichen Speicher-Bedarf von Web Objekten in Betracht und fällt seine Ersetzungsentscheidungen auf Basis deren Größe und deren Zugriffsfrequenz. Dies macht insbesondere Sinn, da durch das Entfernen eines speicherintensiven Objekts Platz für mehrere kleine Objekte geschaffen wird. Dieser Algorithmus wird in der späteren Evaluation die besten Ergebnisse hervorbringen.

4.4 Evaluation der Leistung

In dieser Sektion werden nun Testergebnisse der oben erklärten Mechanismen und Algorithmen gezeigt und diskutiert. Auch werden Vergleiche zu herkömmlichen Methoden aufgezeigt.
Für die Tests wurde ein Prototyp durch die Modifikation eines W3C Jigsaw Java Web Server Version 2.0.5 implementiert [16]. Der GOS und die dafür benötigten Mechanismen wurden dem Jigsaw Server hinzugefügt um Cooperative Caching möglich zu machen. Drei Ersetzungsmechanismen (LFU-Aging, Weighted-LFU und "Global Last Recently Used" (GLRU oben beschrieben als LRU)) wurden auf einem Linux PC Cluster implementiert, um die Effekte von Cooperative Caching in Verbindung mit den Verschiedenen Mechanismen zu evaluieren[1]. Das Zuschalten von Cooperative Caching (CC) bedeutet im Klartext, dass ein Web Server bei einem Content Miss zuerst bei allen anderen Servern im Cluster anfragt, ob einer das benötigte Web Objekt im Cache hält, bevor er dann möglicherweise den Home Node für ein Laden des Objektes von der Datenbank kontaktiert. Ohne CC würde der Server bei einem Content Miss sofort die Datenbank selbst kontaktieren.

Total size	6.756 Gbytes
No. of files	89 689
Average file size	80 912 bytes

[1] Die Leistung wurde anhand eines 32 Maschinen umfassenden Linux PC Cluster gemessen. Bei den Servern handelte es sich um 733MHz starke Maschinen die mit dem Linux 2.2.4 Betriebssystem betrieben wurden. Alle Server waren durch einen 80-port Cisco actalyst 2980G Fast Ethernet Switch verbunden. Während des Tests agierten 16 Maschinen als Clients und der Rest als Server. Die Web Objekte waren auf einem, für alle Server zugänglichen, NFS Server mit zwei 450MHz starken Pentium 3 CPUs unter Linux 2.2.4 gespeichert.

Abbildung 6 - Zusammenfassung der Datencharakteristika

Es gibt neben der Verwendung unterschiedlicher Ersetzungsalgorithmen zwei Möglichkeiten die Leistung des Cache des Clusters zu verändern:

- Die Skalierung der Clustergröße
- Die Skalierung der Cachegröße jedes Web Servers

Die Skalierung der Clustergröße wurde in der Form umgesetzt, dass alle Web Server mit einer Speichergröße von 64 Megabyte ausgestattet wurden und deren Anzahl von zwei auf sechzehn erhöht wurde.

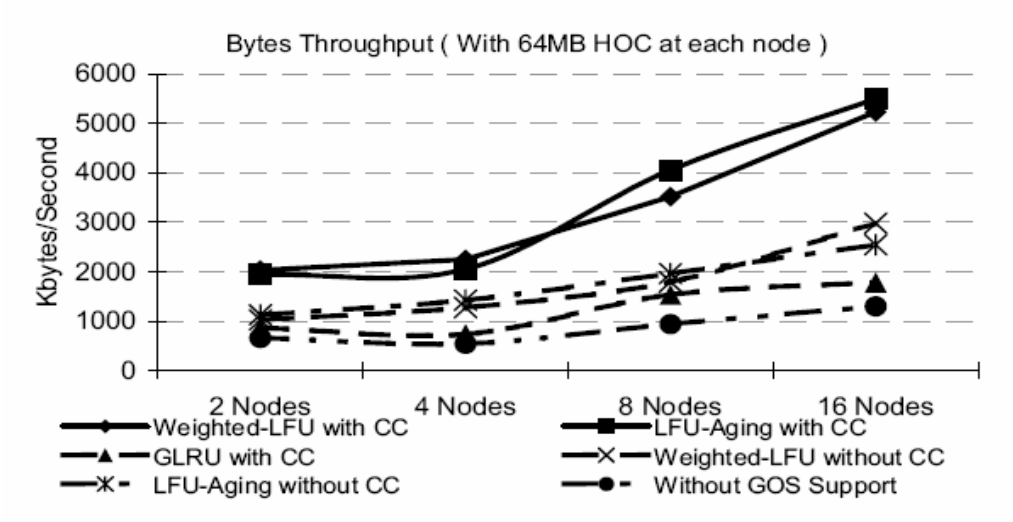

Abbildung 7 - Effekte der Skalierung der Clustergröße

Die Kurven (Abb.7) zeigen deutlich, dass CC in Verbindung dem Weighted-LFU die Leistung des Web Server Clusters deutlich erhöht. Im Fall mit sechzehn Web Servern liegt der Anfragendurchsatz 4,59-mal höher als ohne CC. Dieser Faktor erhöht sich jedoch weiter, wenn man dem System noch mehr Web Server anschließen würde. Zusätzlich lässt sich anhand des Graphen ablesen, dass der Weighted-LFU Algorithmus zu wesentlich besseren Ergebnissen führt, als der LRU Algorithmus.

Die Skalierung der Cachegröße eines jeden Web Servers im Cluster hat zur Folge, dass sich auch die Gesamtgröße des GOS ändert. Während des Tests wurde die “Global Cache Hit Rate“ (GCHR) eines Clusters mit sechzehn Web Servern untersucht. Die GHCR bedeutet, dass ein angefragtes Web Objekt sich im GOS, also entweder im eigenen Cache eines Servers, oder in dem eines anderen Servers im Cluster befindet. Die Größe der Speicher in den einzelnen Servern wurde über den Zeitverlauf erhöht. Beginnend bei 8 Megabyte, was einer relativen Größe des GOS von 1,8% der gesamten Datenmenge entspricht (vgl. Abb.8), bis hin zu 64 Megabyte, was einem relativen Größe des GOS von 14,4% gleichkommt. Dieses Verhältnis wird als relative Cachegröße bezeichnet.

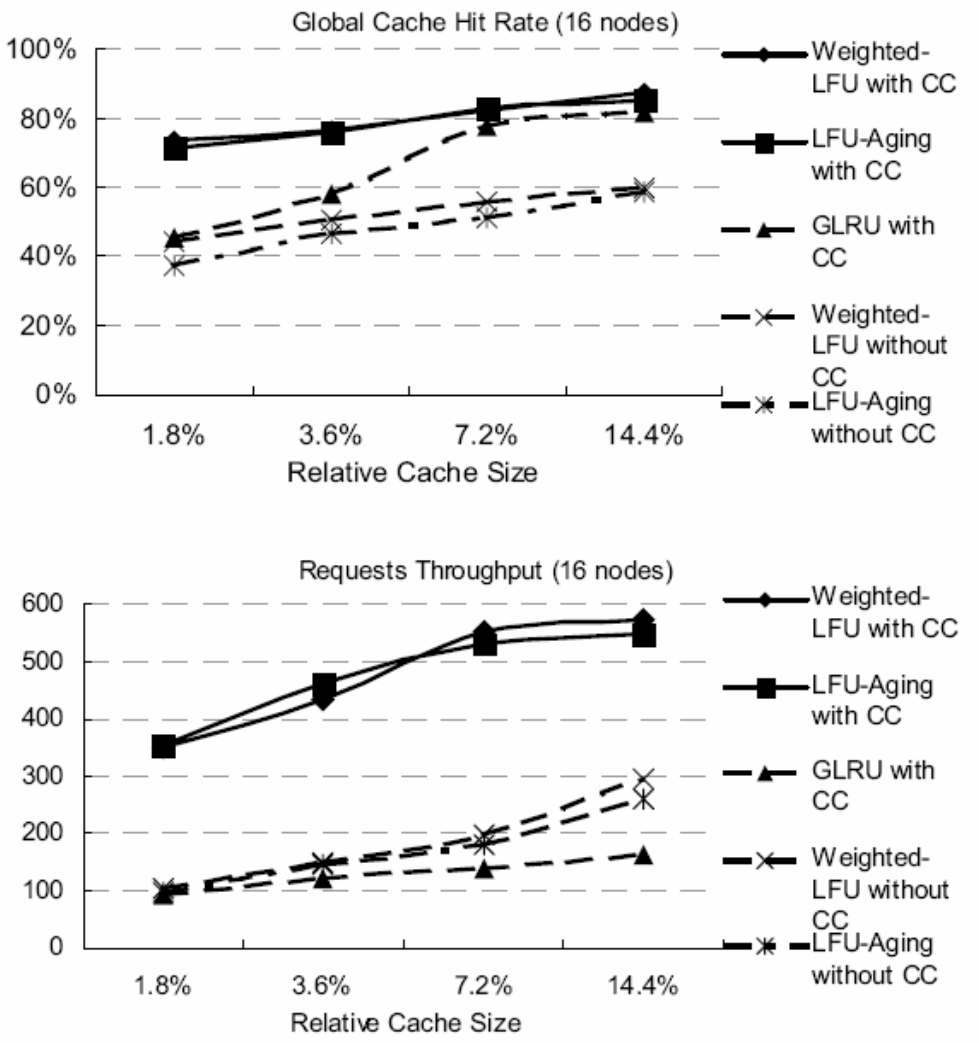

Abbildung 8 - Effekte der Skalierung der Cachegröße

In diesem Zusammenhang wird vor allem der Wert für das “Cooperative Caching“ mit dem Weighted-LFU Algorithmus betrachtet. Zu sehen ist (Abb.8), dass sich mit zunehmender relative auch die Anzahl der abgearbeiteten Anfragen (“Request Throughput“) erhöht. Dies geschieht allerdings nicht linear. So führt beispielsweise eine Cachegröße von 2 Megabyte je Web Server zu einer “Global Hit Rate“ (GHR) von 73%. Erhöht man die relative Cachegröße auf 14,4%, also 16 Megabyte je Web Server, so kommt man sogar auf eine GHR von 87%. Dieser Wert ist bereits beträchtlich, da sich die Anzahl der bearbeitbaren Anfragen pro Sekunde von 350 auf 550 erhöht hat. Das entspricht einer Steigerung von 70%. Algorithmen ohne CC steigern die Anzahl ihrer Anfragebearbeitungen pro Sekunde um nahezu 200% mit einer Vergrößerung der relativen Cachegröße. Jedoch sind deren Ergebnisse bei einer kleinen relativen Cachegröße so schlecht, dass dies eine solche Steigerung erst möglich macht. Algorithmen mit CC liegen, was den Anfragendurchsatz betrifft immer zwischen dem 2-fachen und dem 3,5-fachen von Algorithmen ohne CC.
zusätzlich lässt sich erkennen, dass beide LFU Algorithmen mit CC wesentlich bessere Ergebnisse hervorbringen als ein LRU Algorithmus mit

CC. Während der LRU Algorithmus bei einer Vergrößerung der relativen Cachegröße des Clusters bei der GHR noch zu der Leistung der LFU Algorithmen aufschließen kann, so ist seine Leistung was die Anfragenbearbeitung betrifft sogar noch schlechter als ganz ohne CC.
Die Ursache der Leistungsunterschiede zwischen den beiden LFU Algorithmen bei verschiedenen Cachegrößen liegen darin, dass die Größe beim
Weighted-LFU eine entscheidende Rolle spielt.

4.5 Verwandte Arbeiten

Das Design und die Implementierung von clusterbasierten Web Servern gehört zu den aktuellsten Themen der Internet Forschung. Die meisten Entwicklungsarbeiten beschäftigen sich momentan mit der Anfrageverteilung eines Clusters. Nur wenige Projekte konzentrieren sich auf kooperative Operationen mit Cooperative Caching. Es gibt derzeit, außer dem bereits vorgestelltem modifizierten Jigsaw Server vier interessante Projekte in diesem Themenbereich:

- **“Distributed Cooperative Apache“** [17] (DC Apache)
  Der DC Apache modifiziert dynamisch Hyperlinks in HTML Dokumenten um Client Anfragen zu mehreren kooperativen Web Servern weiterzuleiten. Der Web Server wird Web Objekte direkt zu anderen kooperativen Web Servern migrieren oder kopieren. Um zukünftige Anfragen zum richtigen Server weiterzuleiten, muss der Server in einem migrierten Dokument alle Hyperlinks abändern.

- **“Location Aware Request Distribution“** [18] (LARD)
  Der LARD Mechanismus verteilt Anfragen unter Server Maschinen in einer Cluster Umgebung. Dieser Dispatcher verwaltet eine Liste die Web Objekte mit Web Servern verknüpft. Ankommende Anfragen werden so immer an den Web Server geleitet, mit dem das angeforderte Web Objekt verknüpft ist. Es wird hierbei nur eine sehr eingeschränkte Kommunikation der Web Server untereinander erlaubt.

- **Die Arbeit am “KAIST“** [19]
  Die Entwickler des “Korea Advanced Institute for Science and Technology“ schlagen ein verteiltes Dateisystem mit Cooperative Caching für die Unterstützung Clusterbasierter Web Server vor, welches die LARD Anfragenverteilung nutzt. Zusätzlich soll ein “Duplicate Copy First“ Ersetzungsalgorithmus verwendet werden, der Sicherstellt, dass zuerst ein Objekt aus einem Cache geworfen wird, dass noch an anderer Stelle im GOS präsent ist.

- **“WhizzBee“** [20]
  Der “Whizz Technology’s Whizzbee Web Server“ ist ein Clusterbasierter Web Server der Cooperative Caching realisiert. WhizzBee benutzt die eigens entwickelte “Extensible Cooperative Caching Engine“, die vergleichbar wie der Cooperative Caching Mechanismus arbeitet. Whizzbee fährt einen zentralisierten “Cache-

Load Server“ auf einem Dispatcher, um ankommende Anfragen optimal zu verteilen beziehungsweise festzustellen, ob sich das angeforderte Objekt überhaupt im GOS befindet.

5 Zusammenfassung und Ausblick

Wie bereits eingangs erwähnt ist das wichtigste Maß, um die Leistung eines Web Servers beziehungsweise mehrerer Server in einem Cluster zu messen die mittlere Bearbeitungszeit einer Anfrage. Wenn das Laden eines Web Objektes aus einer möglicherweise räumlich entfernten Datenbank erst zum Zeitpunkt der ankommenden Anfrage erfolgt, erhöht dies deren Bearbeitungszeit erheblich.
Aus diesem Grund macht sich das Zwischenspeichern oft angefragter Web Objekte bezahlt, da so Anfragen wesentlich schneller bearbeitet werden können. Da der Cache eines einzelnen Web Servers oft nicht ausreichend groß ist, um alle nötigen Web Objekte zwischenzuspeichern, fasst man die Caches aller dem Cluster angehörenden Web Server zu einem globalen Cache zusammen. Dies macht aus dem Grund Sinn, da die Netzwerkverbindung zwischen den im Cluster hängenden Servern im Normalfall deutlich schneller als deren Datenbankanbindung ist. Durch diese Zusammenlegung können wesentlich mehr Web Objekte in diesem gemeinsamen Cache gehalten werden, was zu einer Reduzierung der Fernzugriffe führt. Die so verbesserte “Hit Rate“ führt zu einer deutlichen Verbesserung der Antwortzeit.
Alle Tests haben gezeigt, dass der Einsatz von Cooperative Caching die Leistung eines Web Server Clusters erheblich steigert. Mit nur einem kleinen Speicherbereich, welcher für das Cachen verwendet wird, lässt sich eine sehr hohe “Global Cache Hit Rate“ erzielen, wenn die richtigen Cachemechanismen verwendet werden. Der gesamte Cachespeicher eines Clusters wird durch Cooperative Caching wesentlich effizienter genutzt. Das Treffen der besseren Ersetzungsentscheidung in den lokalen Speichern mithilfe von globalen Informationen wird so wesentlich verbessert. Die globalen Zugriffsstatistiken haben sich in diesem Zusammenhang als gute Indikatoren für die “Gefragtheit“ eines Web Objekts bewährt.
Genug Raum für Verbesserungen bleibt im Hinblick auf die Entwicklung eines effizienten Mechanismus für das Weiterleiten eines Web Objekts von einem Web Server zu einem anderen im Cluster in dessen “Hot Object Space“. Ebenso gibt es noch Verbesserungsmöglichkeiten was das Validieren beziehungsweise das Verwalten von dynamischen Web Objekten betrifft.
Cooperative Caching lässt sich jedoch nicht nur auf Clusterbasierte Web Servern einsetzen, sondern auch auf Clusterbasierten Proxy Servern, die viele Eigenschaften mit den Web Servern gemeinsam haben.
Ziel aller zukünftigen Entwicklungen wird in jedem Fall die Minimierung der Antwortzeit für wohl noch leistungsstärkere Anwendungen sein.

Literaturverzeichnis

[1] World Wide Web, **“W3C: The World Wide Web Consortium”**, http://www.w3.org/

[2] Web Services Description Language, **“Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language”**, http://www.w3.org/TR/wsdl20/

[3] Universal Description, Discovery and Integration, **“OASIS § UDDI: UDDI Technical White Paper“**, http://www.uddi.org/pubs/the_evolution_of_uddi_20020719.pdf

[4] Simple Object Access Protocoll . **“W3C Arcitecture Domain Web Services: XML Protocol Working Group“,** http://www.w3.org/2000/xp/Group/

[5] Hyper Text Transport Protocoll, **“W3C Architecture Domain: HTTP – Hypertext Transfer Protocol”**, http://www.w3.org/Protocols/

[6] Extensible Markup Language, **“W3C Architecture Domain: Extensible Markup Language (XML)”**, http://www.w3.org/XML

[7] Business Process Execution Language for Web Services, **“IBM DeveloperWorks Specification: Business Process Execution Language for Web Services Version 1.1”**, http://www-106.ibm.com/developerworks/library/ws-bpel/

[8] Apache Web Server, **“Apache HTTP Server Project”**, http://httpd.apache.org/

[9] Transport Control Protocoll, **“TCP/IP Illustrated, Volume 1: The Protocols”**, von Richard Stevens, Addison-Wesley 1994

[10] HTTP 1.1 Status Code Definitions, **“Hypertext Transfer Protocol RFC 2616”**, von Fielding et al., http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[11] JavaServer Pages, **“SUN: J2EE JavaServer Pages Development”**, http://java.sun.com/products/jsp/

[12] PHP, **“PHP: Hypertext Processor”**, http://www.php.net/

[13] CGI, **“The Common Gateway Interface”**, http://hoohoo.ncsa.uiuc.edu/cgi/

[14] Active Server Pages, **“Microsoft: ASP.Net”**, http://www.asp.net

[15] **“A Scalable Cluster-based Web Server with Cooperative Caching Support”**, von G. Chen, C.L. Wang and F.C M. Lau, University of Hong Kong 2002

[16] **“p-Jigsaw: a cluster-based Web server with cooperative caching support“**, von C. Chen, C.L. Wang and F.C.M. Lau, University of Hong Kong 2003

[17] Distributed Cooperative Apache, **“DC Apache-Module”**, http://www.cs.arizona.edu/dc-apache/apachemodule.htm

[18] Locality-aware request distribution in cluster-based network servers, **“Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)”**,von Vivek SP et al, San Jose CA 1998.

[19] Arbeit am KAIST, **“Efficient cooperative caching for file systems in cluster-based Web servers. Proceedings IEEE International Conference on Cluster Computing”**. von Ahn W, Park S, Park D, Chemnitz Germany 2000.

[20] Whizz Technology’s Whizzbee Web Server, **“Whizz Technology”**, http://www.whizztech.com