System Engineering & Management

Caching - memcached & squid

Autoren:

Alfred Brose
Sebastian Holder
Daniel Kuhn
Sebastian Stadtrecher
Christof Strauch

Seminararbeit:

System Engineering & Management
Wintersemester 2009/2010

Hochschule der Medien, Stuttgart
Studiengang:
Computer Science and Media

Inhalte

1. Einleitung..... 4
1.1 Umfeld und Zielsetzung..... 4
1.2 Ultrahochskalierende Systeme und Caching..... 4
2. Memcached..... 5
2.1 Historie und Relevanz..... 5
2.2 Verwendung..... 6
2.3 Funktionsweise..... 7
2.4 Weitere Hinweise zur Funktionsweise von memcached..... 8
2.5 Begrenzungen von memcached..... 8
2.6 memcached und MediaWiki..... 10
3. memcached-Versuche..... 13
3.1 Eingesetzte Tools..... 13
3.2 Versuchsaufbau..... 16
3.3 Fragestellungen..... 17
3.4 Vorversuche..... 18
3.5 Versuche..... 18
3.6 Erkenntnisse aus den memcached-Versuchen..... 32
4. Squid..... 34
4.1 Einführung..... 34
4.2 Konfiguration..... 35
5. Squid-Versuche..... 37
5.1 Versuchsbeschreibung..... 37
5.2 Test 1: ohne Squid-Cache, ohne MySQL Proxy..... 37
5.3 Test 2: mit vorgewärmtem Squid-Cache, ohne MySQL Proxy..... 37
5.4 Test 3: mit vorgewärmtem Squid-Cache, mit MySQL Proxy..... 37
5.5 Log-Auszüge..... 38
5.6 Log-Auszug..... 39
5.7 Erkenntnisse aus den Squid-Versuchen..... 39
6. Fazit..... 41

Abbildungen

Abbildung 1: MediaWiki – Cache-Objekte..... 11
Abbildung 2: memcached-Versuche - Netzwerkdiagramm..... 17
Abbildung 3: memcached 1 - Items Cached..... 19
Abbildung 4: memcached 1 - Cache Hits and Misses..... 19
Abbildung 5: memcached 1 - Network Traffic (Bits/sec)..... 19
Abbildung 6: Server 2 - Cache Hits and Misses..... 19
Abbildung 7: Server 2 - Current Connections..... 19
Abbildung 8: memcached 2 - Network Traffic..... 20
Abbildung 9: memcached 2 - Items Cached..... 20
Abbildung 10: memcached 2- Requests/sec..... 20
Abbildung 11: memcached 3 - Current Connections..... 21
Abbildung 12: memcached 3 - Cache Hits and Misses..... 21
Abbildung 13: memcached 3 - Items Cached..... 21
Abbildung 14: memcached 3 - Network Traffic..... 21

Abbildung 15: Wikipedia, Requests/sec..... 21
Abbildung 16: memcached 1 - Cache Hits and Misses..... 23
Abbildung 17: memcached 1 - Current Connections..... 23
Abbildung 18: memcached 1 - Items Cached..... 23
Abbildung 19: memcached 1 - Network Traffic (Bits/sec)..... 23
Abbildung 20: memcached 1 - Requests..... 24
Abbildung 21: memcached 2 - Current Connections..... 24
Abbildung 22: memcached 2 - Cache Hits and Misses..... 24
Abbildung 23: memcached 2 - Items Cached..... 24
Abbildung 24: memcached 2 - Network Traffic (Bits/sec)..... 24
Abbildung 25: memcached 2 - Requests/sec..... 24
Abbildung 26: memcached 3 - Cache Hits and Misses..... 25
Abbildung 27: memcached 3 - Current Connections..... 25
Abbildung 28: memcached 3 - Items Cached..... 25
Abbildung 29: memcached 3 - Network Traffic (Bits/sec)..... 25
Abbildung 30: memcached 3 - Requests/sec..... 25
Abbildung 31: memcached 1 - Cache Hits and Misses..... 27
Abbildung 32: memcached 1 - Items Cached..... 27
Abbildung 33: memcached 1 - Requests..... 27
Abbildung 34: memcached 2 - Items Cached..... 28
Abbildung 35: memcached 2 - Cache Hits and Misses..... 28
Abbildung 36: memcached 2 - Requests/sec..... 28
Abbildung 37: memcached 3 - Cache Hits and Misses..... 28
Abbildung 38: memcached 3 - Items Cached..... 28
Abbildung 39: memcached 3 - Requests/sec..... 28
Abbildung 40: memcached 1 - Cache Hits and Misses..... 30
Abbildung 41: memcached 1 - Requests..... 30
Abbildung 42: memcached 2 - Cache Hits and Misses..... 31
Abbildung 43: memcached 2 - Requests/sec..... 31
Abbildung 44: memcached 3 - Cache Hits and Misses..... 31
Abbildung 45: memcached 3 - Requests/sec..... 31

Tabellenverzeichnis

Tabelle 1: memcached - Operationen zum Setzen von Schlüssel-Wert-Paaren.....7
Tabelle 2: MediaWiki – Konfigurationsparameter zur Verwendung von memcached..... 11
Tabelle 3: MySQL Proxy – Erläuterung der Aufrufparameter..... 14
Tabelle 4: memcached-Versuche – Netzwerkknoten und darauf betriebene Server-Dienste.....16
Tabelle 5: memcached Test 1 – memcached nicht vorgewärmt, kein MySQL Proxy..... 19
Tabelle 6: memcached Test 2 – memcached vorgewärmt, kein MySQL Proxy.....23
Tabelle 7: memcached Test 3 – memcached nicht vorgewärmt, mit MySQL-Proxy.....27
Tabelle 8: memcached Test 4 – memcached vorgewärmt, mit MySQL-Proxy.....30
Tabelle 9: memcached – Vergleich der Messungen..... 33
Tabelle 10: Squid Test1 – ohne Squid-Cache, ohne MySQL-Proxy..... 37
Tabelle 11: Squid Test 2– mit vorgewärmtem Squid, ohne MySQL-Proxy.....37
Tabelle 12: Squid Test 3 – vorgewärmter Squid-Cache, mit MySQL-Proxy..... 38
Tabelle 13: memcached und Squid - Gegenüberstellung aller Tests.....41

1. Einleitung

1.1 Umfeld und Zielsetzung

Das vorliegende Dokument beschreibt die Durchführung einiger Versuche im Rahmen des Seminars „System Engineering & Management“, Wintersemester 2009/2010, sowie die daraus resultierenden Erkenntnisse. Grundlegende Zielsetzung des Seminars war es, den Betrieb ultrahochskalierbarer Webseiten am Beispiel der Wikipedia-Architektur praktisch nachzuvollziehen und zu simulieren. In mehreren Teams wurden dazu unterschiedliche praxisrelevante Aspekte beleuchtet.

1.2 Ultrahochskalierende Systeme und Caching

Die Versuche und Ergebnisse zum Thema Caching sollen an dieser Stelle beschrieben und ausgewertet werden. Das Zwischenspeichern generierter und mehrfach auszuliefernder Inhalte spielt in vielen Anwendungsbereichen der Informatik eine wichtige Rolle – so auch für den Betrieb einer Webseite mit außergewöhnlich hohen Zugriffszahlen.

Konkret betrachtet werden zwei unterschiedliche Caching-Technologien: Memcached für das Zwischenspeichern einzelner „Objekte“, sowie Squid zum Caching gesamter auszuliefernder Seiten. Zu beiden Technologien werden an entsprechender Stelle detailliertere Einführungen gegeben.

Sowohl memcached als auch Squid werden im Betrieb von Wikipedia erfolgreich eingesetzt, um die zugrundeliegende Plattform MediaWiki zu skalieren[1,2].

1 vgl. http://www.nedworks.org/~mark/presentations/san/Wikimedia%20architecture.pdf

2 vgl. http://www.scribd.com/doc/50055/workbook2007

2. Memcached

In diesem Abschnitt soll die Caching-Technologie memcached eingeführt werden. Sofern nicht anders vermerkt, entstammen die wiedergegebenen Informationen der memcached-Website[3] sowie insbesondere den FAQ des Projektes[4].

Bei memcached handelt es sich um ein verteiltes Objektcaching-System im Arbeitsspeicher. Dazu wird der Dienst/Daemon memcached auf einem oder mehreren Knoten gestartet und einer Applikation bekannt gemacht. Diese kann Schüssel-Wert-Paare auf memcached-Knoten abspeichern und von diesen lesen. memcached stellt damit einen verteilten, flüchtigen Key-Value-Speicher, der für diverse Einsatzzwecke geeignet ist; typischerweise wird memcached verwendet, um dynamische Web-Anwendungen zu beschleunigen, indem Datenbank-Inhalte (gegebenenfalls in aufbereiteter Form) in memcached gespeichert und auf diese Weise die Datenbank-Server entlastet werden.

2.1 Historie und Relevanz

memcached wurde ursprünglich im Rahmen Performance-verbessernder Maßnahmen für die Website LiveJournal.com entwickelt. Zur Zeit der Entwicklung im Jahr 2003 wurden bei LiveJournal pro Tag mehr als 20 Millionen dynamisch generierte Seiten von rund einer Millionen Nutzern abgefragt, die von einer Reihe von Web- und Datenbank-Servern ausgeliefert wurden. „memcached dropped the database load to almost nothing, yielding faster page load times for users, better resource utilization, and faster access to the databases on a memcache miss.“[4]

memcached wird seit seiner Entwicklung in diversen hochskalierenden Plattformen eingesetzt, etwa YouTube, Twitter, Digg, Flickr oder auch der Wikipedia. Ebenso bieten Content Management Systeme wie Drupal, Blogsysteme wie Wordpress oder auch Web Applikationsframeworks wie Ruby on Rails direkte Unterstützung für memcached an[3,5,6]. Ebenso verhält es sich mit der im Rahmen dieses Masterseminars betrachteten Wiki-Software MediaWiki, die aufgrund ihrer Historie als Wikipedia-Plattform diverse Caching-Verfahren, zu denen insbesondere memcached zählt, direkt unterstützt (siehe dazu 2.6.3.). Gemäß den Ausführungen von Bergsma und Mituzas verwendet die Wikipedia als größte Installation von MediaWiki memcached als wesentliche Technologie zum Caching von Objekten[1,2]. Wenngleich neuere Entwicklungen wie NoSQL-Datenbanken mit eigenen Indizierungs- und Caching-Verfahren den Einsatz von memcached in Zukunft möglicherweise verringern werden[7], so ist für hochskalierende Websites mit klassischen relationalen Datenbanken der Einsatz von memcached als Objektcache häufig alternativlos. Insbesondere im Zusammenspiel mit MySQL als Datenbank-Backend lassen sich deutliche Performance-Verbesserungen durch den Einsatz von memcached erzielen, wie sie etwa das MySQL Peformance Blog empirisch ermittelt hat[8]. Hoff fasst die Ergebnisse dieser Messungen bezüglich MySQL und memcached wie folgt zusammen: „When memcached has enough memory (so records being accessed are in RAM), memcached + MySQL can provide a 10x performance boost over MySQL alone.“[9]

3 vgl. http://www.danga.com/memcached/
4 vgl. http://code.google.com/p/memcached/wiki/FAQ
5 vgl. http://highscalability.com/blog/category/memcached, insbesondere
6 vgl. http://de.wikipedia.org/wiki/Memcached#Software.2C_die_memcached_verwendet
7 vgl. http://highscalability.com/blog/2010/2/26/mysql-and-memcached-end-of-an-era.html
8 vgl. http://www.mysqlperformanceblog.com/2009/10/15/mysql-memcached-or-nosql-tokyo-tyrant-part-1/
vgl. http://www.mysqlperformanceblog.com/2009/10/16/mysql_memcached_tyrant_part2/
vgl. http://www.mysqlperformanceblog.com/2009/10/19/mysql_memcached_tyrant_part3/
9 vgl. http://highscalability.com/blog/2009/10/28/and-the-winner-is-mysql-or-memcached-or-tokyo-tyrant.html

2.2 Verwendung

2.2.1. memcached-Server

Um memcached verwenden zu können, wird der ausgelieferte Daemon zunächst mittels eines einfachen Shell/Commandline-Aufrufs auf einem oder mehreren Rechnern gestartet:

```
# ./memcached -d -m 2048 -l 10.0.0.40 -p 11211
```

Die obige Zeile startet memcached als Daemon (`-d`), der maximal 2GB (`-m 2048`) Arbeitsspeicher verwenden darf und eine Bindadresse (`-l 10.0.0.40`) sowie eine Portangabe (`-p 12211`) übergeben bekommt.

2.2.2. Client-APIs

Um Anwendungen Zugriff auf memcached-Server zu gewähren, stellt das Projekt APIs für diverse Programmiersprachen und Plattformen bereit[10]. Da memcached die Datenstruktur eines Dictionaries / einer Map implementiert, können mittels der Client-APIs Schlüssel-Wertpaare gesetzt, Werte anhand eines Schlüssels abgefragt, sowie optional Schlüssel-Wert-Paare wiederum entfernt werden.

Wird memcached zum Caching von Datenbank-Inhalten eingesetzt, läuft eine typische Interaktion zwischen einer Applikation und memcached wie folgt ab:

1. Abfrage eines Datensatzes aus memcached mittels Schlüssel
2. Falls in memcached kein Wert zum Schlüssel gefunden wurde:
    1. Objekt aus der Datenbank abfragen
    2. Gegebenenfalls Modifikationen oder Transformationen des Objekts vornehmen
    3. Objekt unter Schlüssel in memcached ablegen
3. Falls Objekt geändert wird:
    1. Geändertes Objekt in die Datenbank schreiben
    2. Geändertes Objekt in memcached ablegen

Ein Beispiel für einen typischen lesenden Zugriff in der Sprache PERL zeigt folgendes Code-Snippet:

```
sub get_foo_object {
   my $foo_id = int(shift);
   my $obj = '''$::MemCache->get("foo:$foo_id");'''
   return $obj if $obj;

   $obj = $::db->selectrow_hashref("SELECT .... FROM foo f, bar b ".
                                   "WHERE ... AND f.fooid=$foo_id");
   '''$::MemCache->set("foo:$foo_id", $obj);'''
   return $obj;
}
```

10 Perl, C, C#, PHP, Python, Java, Ruby und weitere, siehe http://code.google.com/p/memcached/wiki/Clients. Sofern keine Client-API für die gewünschte Programmiersprache verfügbar ist, kann eine solche anhand der memcached Protokoll-Spezifikation auch selbst entwickelt werden.

Da memcached kein Locking unterstützt, können bei konkurrierendem Zugriff auf gleiche Schlüssel etwa Lost-Updates auftreten, zum Beispiel in folgendem Szenario:

1. Client 1 liest ein Objekt aus memcached und verändert es
2. Client 2 liest ein Objekt aus memcached, bevor Client 1 seine Änderungen nach memcached zurückgeschrieben hat
3. Client 1 schreibt das veränderte Objekt in memcached
4. Client 2 schreibt das von ihm gelesene (und gegebenenfalls veränderte) Objekt nach memcached

In Schritt 4 überschreibt Client 2 somit die Änderungen von Client 1. Memcached stellt keine Möglichkeiten des Lockings bereit, um derartige Race-Conditions zu verhindern. Dies ist in der Applikationslogik zu berücksichtigen. Helfen können dabei gemäß der memcached-Dokumentation unterschiedliche Operationen zum Setzen von Objekten in memcached

Operation	Erläuterung
`set`	Setzt ohne weitere Prüfung einen Wert für einen Schlüssel
`add`	Fügt einen Wert unter einem Schlüssel nur dann hinzu, falls unter dem Schlüssel zuvor kein Wert abgelegt war
`replace`	Überschreibt einen Wert unter einem Schlüssel, falls bereits zuvor ein Wert zu diesem Schlüssel abgelegt war.

Tabelle 1: memcached - Operationen zum Setzen von Schlüssel-Wert-Paaren

2.3 Funktionsweise

2.3.1. Mapping von Objekten auf memcached-Instanzen

Um memcached in Applikationen zu verwenden, werden der verwendeten Client-API zunächst die verfügbaren memcached-Server sowie ihre Gewichtung bekannt gemacht. Die Gewichtung der memcached-Server erfolgt anhand der Größe des Arbeitsspeichers, der den Servern zugewiesen wurde.

Beim Lesen und Schreiben von Objekten nach memcached findet ein zweistufiges Hashing-Verfahren über den Schlüssel, unter dem das Objekt gelesen beziehungsweise abgelegt werden soll, statt:

1. Stufe: Ermittlung des Servers, auf dem das Schlüssel-Wert-Paar abgelegt beziehungsweise von dem der Wert gelesen werden soll

Anhand der Anzahl und Gewichtung der konfigurierten Server wird diesen ein Teil des Bildbereichs der Hash-Funktion zugeordnet. Die memcached-API ermittelt denjenigen memcached-Server, in dessen Wertebereich der Schlüssel-Hash fällt.

2. Stufe: Ermittlung des Wertes zum Schlüssel auf dem in Stufe 1 gefundenen Server

Nachdem in Stufe 1 der Server ermittelt wurde, auf dem das Schlüssel-Wert-Paar zum gesuchte Schlüssel zu finden beziehungsweise abzulegen ist, fragt die Client-API diesen Server mit dem gesuchten Schlüssel an. Der memcached-Server schlägt diesen Schlüssel nach und liefert – bei lesendem Zugriff – das darunter abgelegte Objekt (oder bei Nichtexistenz einen null-Wert) zurück beziehungsweise legt – bei schreibendem Zugriff – den übergebenen Wert unter dem angegebenen Schlüssel ab.

Beispielablauf

1. Client 1 möchte unter dem Schlüssel "foo" den Wert "barbaz" setzen
   - Es wird die Liste aller memcached-Server Server (z.B. A, B, C) herangezogen[11]
   - Der Schlüssel "foo" wird gegen diese Server gehasht (`hash("foo") % 3`)
   - Das Resultat sei, dass das Schlüssel-Wert-Paar auf Server B abzulegen ist
   - Client 1 verbindet sich dann zu Server B und setzt den Wert "barbaz" unter dem Schlüssel "foo"
2. Als nächstes möchte Client 2 den Schlüssel "foo" nachschlagen
   - Client 2 verwendet die gleiche memcached-library wie Client 1 und kennt ebenfalls die Server-Konfiguration (Server A, B, C; gleich gewichtet).
   - Client 2 ermittelt durch Hashing des Schlüssels gegen die Server-Liste, dass der Schlüssel auf Server B nachgeschlagen werden kann.
   - Client 2 fragt Server B mit dem Schlüssel "foo" an und erhält den Wert "barbaz" zurück.

2.4 Weitere Hinweise zur Funktionsweise von memcached

- Falls memcached-Knoten nicht erreichbar sind, mappen die memcached-Client APIs die diesen Servern zugeordneten Schlüssel auf die noch verfügbaren memcached-Server.
- Verschiedene Client-APIs können unterschiedliche Hashing-Algorithmen implementieren. Jedoch verhalten sich memcached-Server davon unabhängig stets gleich.
- memcached ist als nicht-blockierender, Event-basierter Server implementiert. Dies dient dazu, dass sogenannte C10K Problem[12] zu lösen. Ferner werden auf diese Weise bessere Skalierungseigenschaften erzielt gegenüber Datenbanken, die bei Schreiboperationen blockieren, gegebenenfalls auf Plattenspeicher zugreifen und möglicherweise zusätzlichen Overhead durch die Realisierung von ACID[13]-EIgenschaften mit sich bringen.
- Hinsichtlich der Cache-Bereinigung agiert memcached als LRU[14]-Cache. Außerdem werden Ablaufdauern für Cache-Objekte unterstützt, die als Werte „für immer“ oder Zeitpunkte in der Zukunft annehmen können.

2.5 Begrenzungen von memcached

memcached ist mit den im Folgenden aufgeführten Begrenzungen behaftet.

11 Die memcached-Server seien in diesem Beispiel der Einfachheit halber gleich gewichtet.
12 vgl. http://www.kegel.com/c10k.html
13 Atomicity, Consistency, Isolation, Durability
14 Least recently used

2.5.1. Größe der speicherbaren Objekte

Die Längen der Schlüssel sind auf 250 Bytes begrenzt, Werte auf 1MB. memcached eignet sich daher nicht, um Mediendateien oder andere große BLOBs[15] abzuspeichern; für diese Anwendungsszenarien sind andere Lösungen wie MogileFS[16] zu bevorzugen.

2.5.2. Geschwindigkeit gegenüber lokalem RAM

Die Verwendung von memcached ist langsamer gegenüber der Verwendung lokalen Arbeitsspeichers. Letzteres ist jedoch wiederum nur durch lokale Applikationen und Clients adressierbar.

2.5.3. Redundanz

Memcached unterstützt keine Redundanz für memcached-Server. Dies bedeutet, dass die memcached-APIs in Applikationen mit der Nicht-Verfügbarkeit konfigurierter memcached-Knoten umgehen können müssen.

2.5.4. Failover

Memcached unterstützt kein automatisches Failover, da es unter den memcached-Servern keine Master gibt. Daher müssen Applikationen auf den Ausfall von memcached-Knoten reagieren, indem sie etwa...

- diese Ausfälle ignorieren.
- nicht-verfügbare Knoten aus der konfigurierten Liste der memcached-Knoten entfernen. (Achtung: diese Reaktion invalidiert den Cache!)
- Hot-Spare-Knoten verwenden, die über die gleiche IP-Adresse wie tote memcached-Knoten erreichbar sind.
- einen Key-Rehashing-Modus verwenden, der in früheren Versionen von memcached standardmäßig verwendet wurde. Dies kann jedoch angesichts der „flapping servers“-Problematik[17] unerwünscht sein.

2.5.5. Sicherheit

Memcached stellt keine Authentisierung oder andere Verfahren zur Absicherung des Caches bereit. Falls memcached-Knoten nicht mittels Authentisierungs-Proxies, Firewalls oder ähnlichem geschützt werden, haben beliebige Clients, die memcached-Knoten über das Netzwerk erreichen können, mittels telnet Vollzugriff auf memcached-Knoten. Diese Belange sind insbesondere zu bedenken, wenn memcached in unsicheren Umgebungen – etwa in geteilten Infrastrukturen ohne eigene administrative Befugnisse (z.B. Shared-Hosting-Umgebungen) – betrieben werden soll.

2.5.6. Atomizität

Das memcached-Protokoll sieht keine Atomizität für Folgen von Kommandos, die an memcached-Knoten abgesetzt werden, vor.

15 Binary Large Objects

16 http://www.danga.com/mogilefs

17 Gemeint sind Server, in diesem Fall memcached-Knoten, die in kurzen Zeitintervallen abwechselnd verfügbar und nicht verfügbar sind.

2.5.7. Betrieb in virtualisierten Infrastrukturen

Virtualisierte Infrastrukturen können gemäß memcached-FAQ die Performance des Caches erheblich beeinträchtigen, da memcached typischerweise eine große Menge Arbeitsspeicher kontrollieren soll und Performance-Einbußen auftreten, wenn dieser Arbeitsspeicher durch Betriebssystem oder Hypervisor auf Plattenspeicher ausgelagert/geswappt wird.

2.6 memcached und MediaWiki

MediaWiki unterstützt memcached seit einer der ersten Versionen (bereits vor Version 1.1) als einen von mehreren möglichen Caching-Mechanismen, der für große Seiten mit hoher Last empfohlen wird[18].

2.6.1. Voraussetzungen

Um memcached mit MediaWiki zu verwenden, sind folgende Voraussetzungen zu erfüllen:

- Der vom Webserver verwendete PHP-Interpreter muss mit der Option `--enable-sockets` kompiliert sein.
- Die Bibliothek libevent[19] muss verfügbar sein.
- Falls memcached unter Linux betrieben wird, ist optional der epoll-rt Patch[20] auf den Linux-Kernel anzuwenden beziehungsweise ein Kernel zu verwenden, der diesen Patch berücksichtigt

memcached-Server können auf einem oder mehreren Knoten im Netzwerk betrieben und von einem oder mehreren Web Servern zugegriffen werden.

2.6.2. Sicherheit

Die MediaWiki-Entwickler sowie die memcached-Projektmitarbeiter weisen auf angemessenen Zugriffsschutz durch Einsatz von Firewalls hin. Falls dies nicht geschieht und memcached-Server für Unbefugte[21] erreichbar sind, können gravierende Folgen auftreten: „An attacker familiar with MediaWiki internals could use this to give themselves developer access and delete all data from the wiki's database, as well as getting all users' password hashes and e-mail addresses.“

2.6.3. Konfiguration

Um memcached als Caching-Strategie für MediaWiki zu verwenden, sind folgende Einstellungen in der zentralen MediaWiki-Konfigurationsdatei `LocalSettings.php` zu treffen:

```
$wgMainCacheType = CACHE_MEMCACHED;
$wgParserCacheType = CACHE_MEMCACHED; # optional
$wgMessageCacheType = CACHE_MEMCACHED; # optional
$wgMemCachedServers = array( "127.0.0.1:11000" );
$wgSessionsInMemcached = true; # optional
```

Die folgende Tabelle erläutert die einzelnen Konfigurationsparameter.

18 Quelle dieses Unterkapitels: http://www.mediawiki.org/wiki/Memcached
19 http://www.monkey.org/%7Eprovos/libevent/
20 http://www.xmailserver.org/linux-patches/nio-improve.html
21 gemeint sind hier alle Rechner, die keinen Zugriff auf memcached benötigen

Konfigurationsparameter	Erläuterung

`$wgMainCacheType`
`$wgParserCacheType`
`$wgMessageCacheType`
	Definiert die zu verwendende Caching-Strategie für Objekte, Parser und Nachrichten aus jeweils einer Menge von Optionen[22,23,24].

`$wgMemCachedServers`	Konfiguration der memcached-Knoten und ihrer Gewichtung[25]. Die Gewichtung erfolgt anhand der Schachtelungstiefe der zugewiesenen Arrays, z.B.:
	`$wgMemCachedServers = array(`
	`"127.0.0.1:11000", # one gig on this box`
	`array("192.168.0.1:11000", 2 ) # two gigs on the other box`
	`);`

`$wgSessionsInMemcached`	Falls diese Option gesetzt wird, verwenden alle Funktionen, die mit der Sitzungsverwaltung zu tun haben, ihre Daten im unter `$wgMainCacheType` konfigurierten Cache anstatt im Dateisystem[26].

Tabelle 2: MediaWiki - Konfigurationsparameter zur Verwendung von memcached

2.6.4. Cache-Objekte

Die folgende Abbildung gibt einen Überblick über die Inhalte von MediaWiki, die im Cache abgelegt werden, sobald der Konfigurationsoption `$wgMainCacheType` einen anderen Wert als `CACHE_NONE` zugewiesen bekommt[27,28].

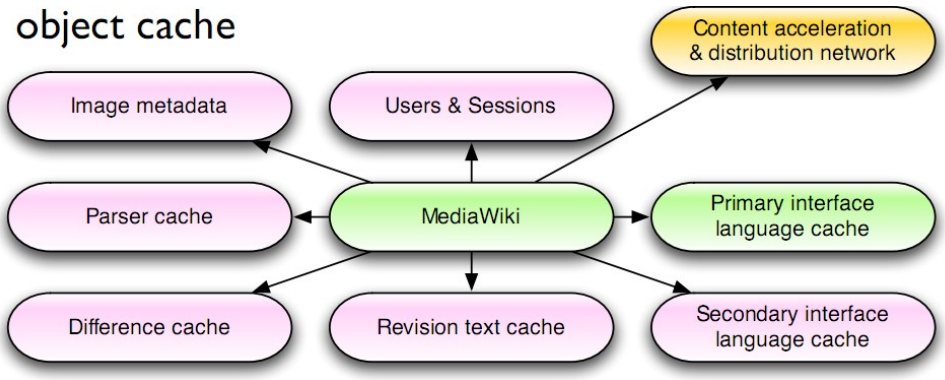

Abbildung 1: MediaWiki – Cache-Objekte

Domas Mituzas macht dazu in seinem Vortrag auf der MySQL Users Conference 2007 folgende Bemerkungen[28]:

- "Parser cache is most critical for application performance - it stores preprocessed HTML output. The content language, wikitext, allows integrating multiple content sources, preprocessing clauses and depend on many other content objects. Page views by users with same settings would not cause re-parsing of content.

22 vgl. http://www.mediawiki.org/wiki/Manual:$wgMainCacheType
23 vgl. http://www.mediawiki.org/wiki/Manual:$wgParserCacheType
24 vgl. http://www.mediawiki.org/wiki/Manual:$wgMessageCacheType
25 vgl. http://www.mediawiki.org/wiki/Manual:$wgMemCachedServers
26 vgl. http://www.mediawiki.org/wiki/Manual:$wgSessionsInMemcached
27 vgl. http://www.nedworks.org/~mark/presentations/san/Wikimedia%20architecture.pdf
28 vgl. http://www.scribd.com/doc/50055/workbook2007

- ‘Compare articles’ output is cached in difference cache, and it is especially efficient with ‘recent changes patrol’ - people reviewing changes.
- Interface language cache combines dynamic interface changes, together with messages from source code. Even parsing the PHP file is quite expensive, so having this serialized or directly in APC cache helps a lot.
- Revision cache was introduced after revision texts (hundreds of gigabytes of content) were moved away from core databases to bigger but slower distributed storage. Some operations do need source text, and hitting the storage backend may be much slower, and having this content in shared cache helps.
- Image metadata keeps simple information about images, that doesn’t have to go to databases, and may be forgotten.
- Session information may be especially dynamic but has no long-term value - writing into database would result with replication pollution, even though on single server it could be quite efficient.
- There’re quite a few other small bits cached, like pre-parsed interface parts, information about database server states, etc.“

3. memcached-Versuche

In diesem Kapitel werden nun die im Rahmen des Seminars durchgeführten Versuche zur memcached-Technologie vorgestellt.

3.1 Eingesetzte Tools

Dieses Unterkapitel stellt zwei Werkzeuge vor, die für die memcached-Versuche verwendet wurden.

3.1.1. Live-Monitoring von memcached-Instanzen mit memcached-top.pl

Auf dem Server 141.62.66.91 wurde ein Perl-Skript[29] zum Live-Monitoring der memcached-Server installiert. Dieses liefert Statistiken über gets, sets, reads, writes und die hit-ratio der einzelnen memcached-Knoten. Das Perl-Skript kann mit verschiedenen Parametern aufgerufen werden:

```
./memcached_top.pl --command
```

Mit dem Parameter `--command` werden detaillierte Informationen über gets, sets, read und writes zum jeweils aktuellen Zeitpunkt zeigt.

Eine kumulierte Statistik der memcached-Zugriffe seit dem Starten der memcached-Server kann mit dem Parameter `--lifetime` angezeigt werden.

```
./memcached_top.pl --lifetime
```

Zu unseren Versuchen zeigen wir jeweils sowohl momentane als auch kumulierte Ausgaben des Tools.

3.1.2. Monitoring der memcached-Server mittels Cacti

Neben dem Live-Monitoring der memcached-Server mittels des oben beschriebenen Tools werden diese zusätzlich mit der Monitoring-Lösung Cacti[30] überwacht. Für Cacti wurde von der Monitoring-Gruppe ein Template[31] zur Überwachung von memcached-Parametern (Cache Hits/Misses, Items cached, Requests/sec, Bytes used etc.) installiert und konfiguriert

3.1.3. MySQL Proxy

Um Effekte der Caching-Technologie memcached zu Provozieren wurde neben dem Erzeugen von Last auch versucht, Netzwerklatenzen zu simulieren. Dazu haben wir zunächst den Datenbank-Server auf einen anderen Netzwerkknoten umgezogen als der, auf dem der Apache-Server betrieben wurde. Um weitere Latenzen zu erzeugen, wurde ein MySQL-Proxy eingesetzt, der über den alle Datenbankzugriffe geleitet wurden und der diese künstlich verzögerte.

29 zu beziehen unter http://code.google.com/p/memcache-top/
30 vgl. http://www.cacti.net/
31 zu beziehen unter http://dealnews.com/developers/cacti/memcached.html

Installation des MySQL Proxys

Für die Versuche wird der Rechner IS-PC11 (141.62.66.111) als MySQL Proxy Server vorbereitet. Dazu wird der MySQL Proxy, der direkt von der MySQL Website als Binary erhältlich ist[32], gestartet.

Konfiguration des MySQL Proxys

Der folgende Aufruf zeigt die Hilfeseite des MySQL Proxy an:

```
mysql-proxy.exe --help-all
```

Gestartet wurde der MySQL Proxy für unsere Versuche wie folgt:

```
mysql-proxy.exe --proxy-backend-addresses=141.62.66.90:3306 --proxy-lua-script=slow-down.lua
```

Die Aufrufparameter haben folgende Bedeutung:

Aufrufparameter	Bedeutung
`--proxy-backend-addresses`	IP und Port des eigentlichen DB Servers
`--proxy-lua-script`	LUA-Skript, das bei jedem Request ausgeführt werden soll

Tabelle 3: MySQL Proxy - Erläuterung der Aufrufparameter

3.1.4. Zugriff auf Proxy und DB

Der Zugriff auf die Datenbank erfolgt über den MySQL-Standardport 3306. Der MySQL Proxy wird so konfiguriert, dass er auf Port 4040 Anfragen akzeptiert.

3.1.5. Verzögernde Skripte

Es wird ein LUA Skript erstellt, das bei jedem Aufruf ausgeführt wird und diesen verzögert. Dazu wird die Funktion `function read_query( packet )` genutzt. Um zusätzlich die Zeit des Requests messen zu können, wird entsprechende Logik in der Funktion `function read_query_result(inj)` hinzugefügt:

```
slow-down.lua
```

```
---
-- getting the query time
--
-- each injected query we send to the server has a start and end-time
--
-- * start-time: when we call proxy.queries:append()
-- * end-time:   when we received the full result-set
--
-- @param packet the mysql-packet sent by the client
--
-- @return
--   * proxy.PROXY_SEND_QUERY to send the queries from the proxy.queries queue
--
function read_query( packet )
```

32 zu beziehen unter http://dev.mysql.com/downloads/mysql-proxy/

```
	io.write("QUERY: ")

	if packet:byte() == proxy.COM_QUERY then
		--print("we got a normal query: " .. packet:sub(2))

		proxy.queries:append(1, packet )

		local result =  proxy.PROXY_SEND_QUERY
		os.execute("\"C:\\Documents and Settings\\praktikum\\Desktop\\mys-
ql-proxy-0.6.0-win32-x86\\doStuff.bat\"")
		return result
	end



end

---
-- read_query_result() is called when we receive a query result
-- from the server
--
-- inj.query_time is the query-time in micro-seconds
--
-- @return
--   * nothing or proxy.PROXY_SEND_RESULT to pass the result-set to the client
--
function read_query_result(inj)
	--print("query-time: " .. (inj.query_time / 1000) .. "ms")
	print("response-time: " .. (inj.response_time / 1000) .. "ms")
end
```

Da ein Aufruf der Art `sleep(milliseconds)` in LUA nicht so leicht möglich ist, wird diese Funktionalität über ein Batch Skript abgedeckt, das lediglich sinnlose Directory-Listings durchführt, damit aber die gewünschte Zeit zubringt.

```
doStuff.bat:
```

```
@echo off
REM this is just bogus to spend some time doing useless stuff.



REM 200ms:
dir /A:-D /B /S "C:\Program Files\OpenOffice.org 3" > nul

REM 50 ms
dir /A:-D /B /S "C:\Program Files\wireshark" > nul
```

3.1.6. Beispiel-Ausgabe des MySQL Proxys während der Versuche

```
QUERY: response-time: 234.375ms
QUERY: response-time: 250ms
QUERY: response-time: 234.375ms
QUERY: response-time: 218.75ms
[...]
```

3.2 Versuchsaufbau

3.2.1. JMeter-Testcase

Für die Messungen wird ein JMeter-Testcase verwendet, der mit 10 parallelen Threads, die in 20 Wiederholungen zufällig acht Artikel aus einer Menge von 93 Wikipedia-Artikeln auswählen. Die Messungen werden von einem Client-Rechner aus durchgeführt. Dieser Rechner wurde über ein Netzwerkkabel am Switch angeschlossen, an dem auch die Server angeschlossen waren.

3.2.2. Netzwerkknoten und Server-Dienste

Die Server-Dienste wurden wie folgt auf die im Rahmen des Seminars verfügbare Infrastruktur verteilt:

Netzwerkknoten	Darauf betriebene Dienste
141.62.66.90	• Mysql • memcached
141.62.66.91	• Apache • memcached
141.62.66.93	• Monitoring-Lösungen (munin, nagios, Cacti) • memcached

Tabelle 4: memcached-Versuche - Netzwerkknoten und darauf betriebene Server-Dienste

Abbildung 2 visualisiert den Versuchsaufbau aus Netzwerk- und Verteilungssicht.

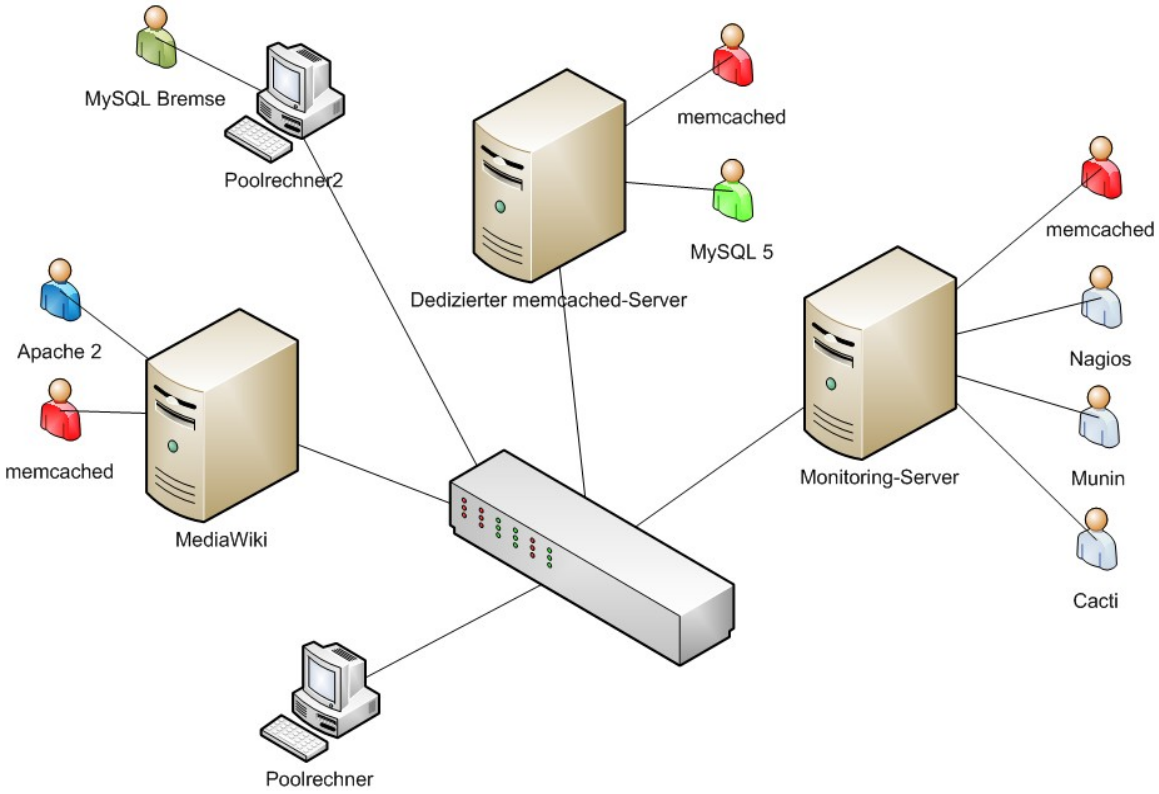

Abbildung 2: memcached-Versuche - Netzwerkdiagramm

3.2.3. Datenbank

Für MySQL wurde der Query-Cache deaktiviert, um Interferenzen mehrerer Caches zu vermeiden. Wo angegeben, wurde der zuvor beschriebene MySQL Proxy gemäß den Einstellungen aus Kapitel 3.1.3. betrieben.

3.2.4. Web Server

Für den Apache Web Server wurden bezüglich des Threadings eine Thread-Anzahl von 18 eingestellt.

3.3 Fragestellungen

Für die memcached-Versuche waren folgende Fragestellungen relevant:

- Können Caching-Effekte durch memcached nachgewiesen werden, indem kürzere Antwortzeiten bei Verwendung von memcached gemessen werden als beim Betrieb ohne memcached?
- Falls nicht: was sind die Gründe dafür?
- Bei welcher Last wird ein Unterschied zwischen dem Betrieb mit oder ohne memcached messbar?
- Welche Vorkehrungen sind im Versuchsaufbau zu treffen, um Caching-Effekte messbar zu machen?

3.4 Vorversuche

Als die ersten Tests durchgeführt wurden, traten einige Probleme mit den Einstellungen und der Hardware auf. In diesen Tests wurde viel Erfahrung mit dem Umgang von memcached gesammelt – insbesondere wurden Fehlerquellen identifiziert.

3.4.1. Probleme

Die Konfiguration der Tests mit JMeter ließ die CPU des Client-PC, der für die Testausführung verwendet wurde, schon bei 20 Threads unter Vollast laufen. Dies hatte zur Folge, dass wir nicht sicher sein konnten ob der Client Rechner die Anfragen und deren Zeitmessung korrekt ermittelt hatte. Zudem wurde erkannt, dass die Serverauslastung bei den Tests ebenfalls bei konstant über 95% lag. Dies war aber – wie sich später herausstellte – ein gutes Zeichen.

Ein weiterer Grund, weshalb wir die ersten Tests nicht als glaubwürdig ansahen, waren Hardware-bedingte Fehler. Bei den Test sind uns einige Geschwindigkeitsschwankungen aufgefallen die uns später zu dem Switch führten, an dem die Server angeschlossen waren. Nach dem Einstecken der Netzwerkkabel am Switch wurde eine Geschwindigkeit von 100Mbit/s angezeigt. Es fiel allerdings auf, dass nach einer unbestimmten Zeit die Bandbreite auf 10MBit/s zurückfiel und auffällig viele RX-Fehler am Netzwerkinterface aufgetreten sind, egal ob der Server ausgelastet wurde oder nicht. Nach einigen Einstellungsversuchen an dem Betriebssystem der Server – welche nichts an dem Fehler änderten – wurden auf Verdacht die Netzwerkkabel getauscht. Nach weiteren Tests stellte sich heraus, dass dies den Fehler behoben hatte. Nach genauerem Betrachten der Kabel wurde schnell klar, warum die Geschwindigkeit gefallen war: die Kabel waren an dem Ende, das zum Server führte, mehrfach mit Isolierband repariert worden.

3.4.2. Effekte

Bei den ersten Tests fielen uns einige Effekte auf:

- Wir testeten anfangs noch mit nur zwei memcached-Servern. Dabei kam es zu einer unerwarteten Verteilung der Objekte auf die memcached-Server. Der memached-Dienst auf dem Web-Server-Knoten (141.62.66.91) erhielt rund 60% der Objekte, der zweite Memcached-Server (141.62.66.90) nur 30%.
- Das Ausführen der Tests aus dem WLAN lieferte unterschiedliche Ergebnisse, daher wurden alle späteren Tests im lokalen kabelgebundenen Netz ausgeführt. Eine direkte Anbindung des Test-Rechners an den Switch der Server führte zu besseren und vor allem zu stabileren Ergebnissen.
- Die Konfiguration des Web-Servers (httpd) ist zudem ausschlaggebend für die Performance.

3.5 Versuche

Nachdem die im vorigen Unterkapitel beschriebenen Probleme behoben waren, führten wir folgende Tests durch, die zu stabilen und reproduzierbaren Ergebnissen führten.

3.5.1. Test 1: memcached nicht vorgewärmt, kein MySQL Proxy

Im ersten Versuch wurden alle memcached-Server vor Beginn des Tests geleert, indem sie neu gestartet wurden. Der MySQL Proxy kam in diesem Versuch nicht zum Einsatz.

JMeter-Messergebnisse

	Proben	Durchschnitt	Mittel	90% Marke	Min	Max	% Fehler	Durchsatz	KB/sek
Gesamt	3200	1103 ms	901	1276 ms	223 ms	134099 ms	0.0	8,01	524,72

Tabelle 5: memcached Test 1 - memcached nicht vorgewärmt, kein MySQL Proxy

Cacti-Graphen

Memcached-Server 1 (141.62.66.93)

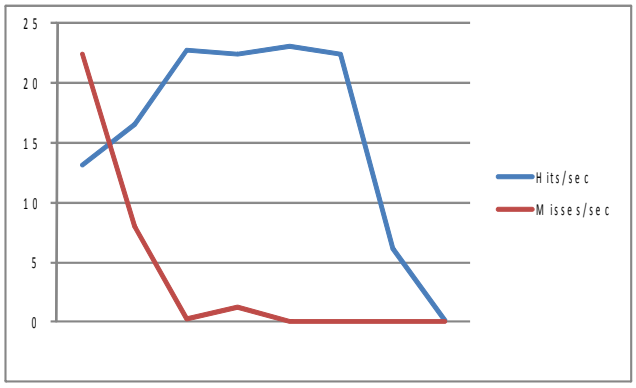

Abbildung 4: memcached 1 - Cache Hits and Misses

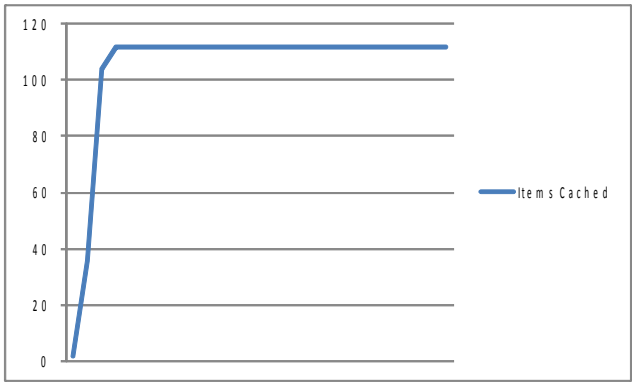

Abbildung 3: memcached 1 - Items Cached

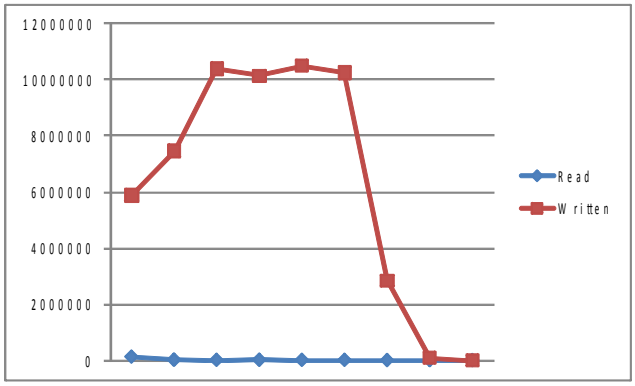

Abbildung 5: memcached 1 - Network Traffic (Bits/sec)

Memcached Server 2 (141.62.66.90)

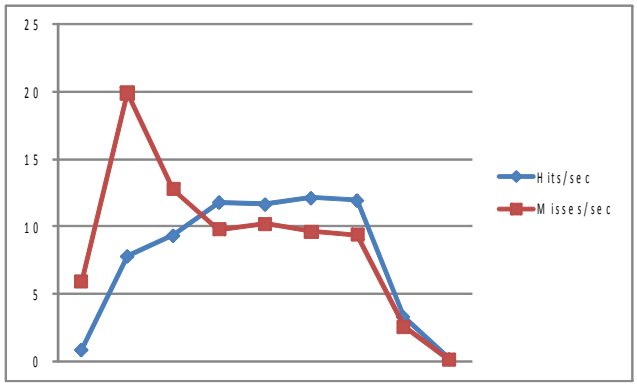

Abbildung 6: Server 2 - Cache Hits and Misses

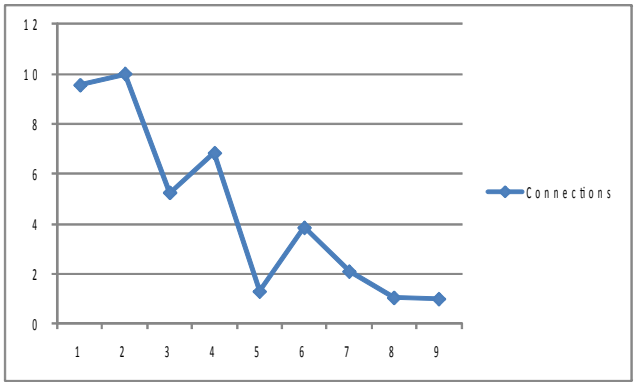

Abbildung 7: Server 2 - Current Connections

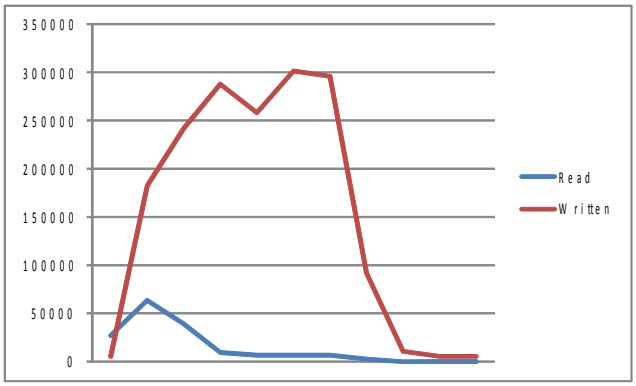

Abbildung 8: memcached 2 - Network Traffic

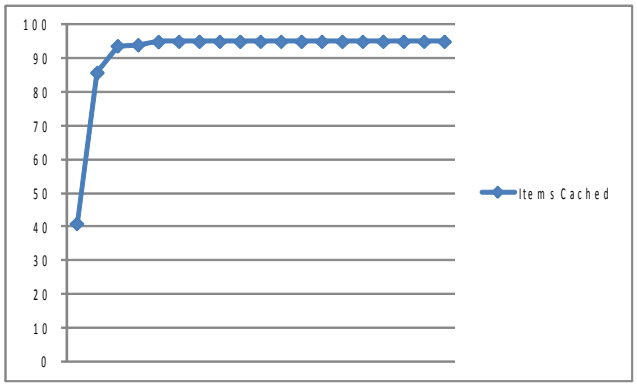

Abbildung 9: memcached 2 - Items Cached

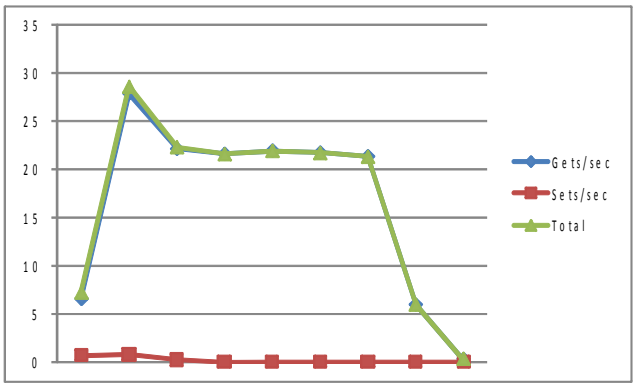

Abbildung 10: memcached 2- Requests/sec

Memcached Server 3 (141.62.66.91)

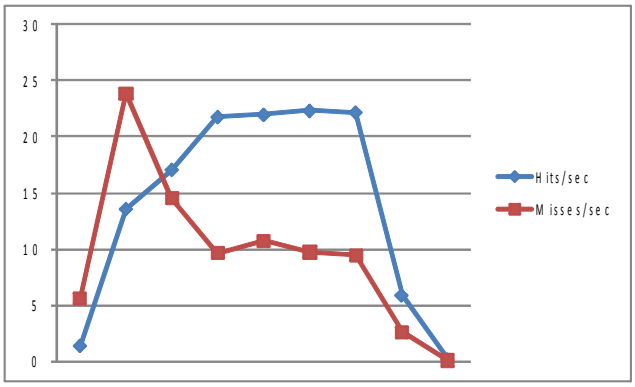

Abbildung 12: memcached 3 - Cache Hits and Misses

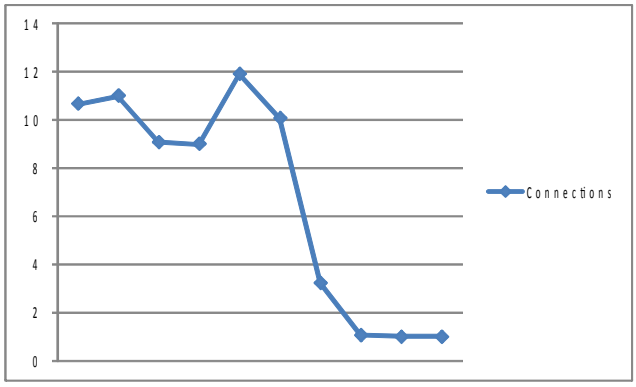

Abbildung 11: memcached 3 - Current Connections

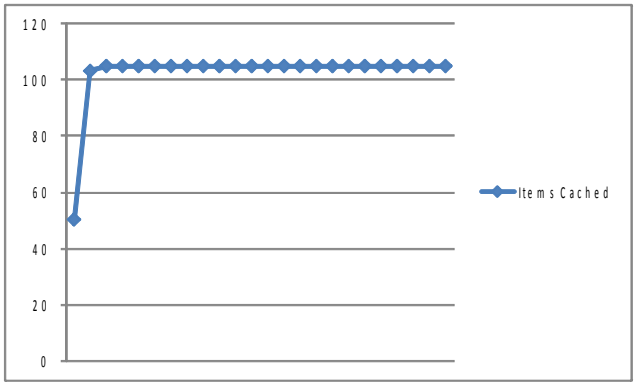

Abbildung 13: memcached 3 - Items Cached

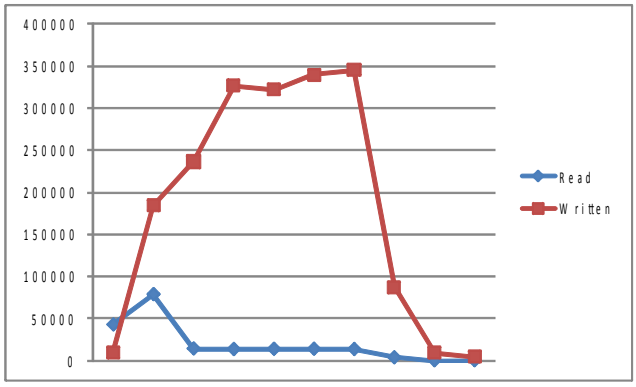

Abbildung 14: memcached 3 - Network Traffic

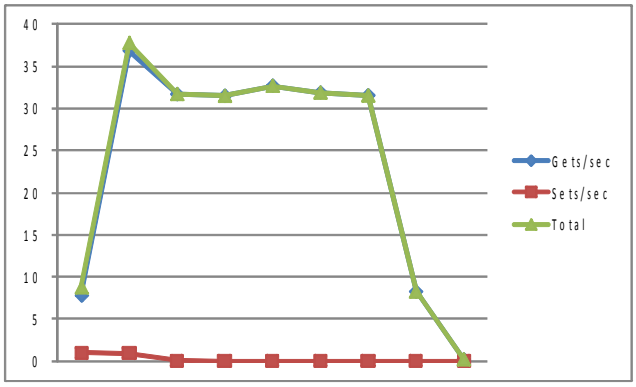

Abbildung 15: Wikipedia, Requests/sec

memcaced-top Ausgaben

Momentaufnahme zu Beginn des Tests:

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE               USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211     0.2%    23.8%   11      40.0ms  0.0     81      1       10.1K   32.1K
141.62.66.90:11211     0.2%    17.0%   8       29.7ms  0.0     58      0       1583    1012
141.62.66.93:11211     1.0%    21.0%   9       83.4ms  0.0     94      3       83.6K   820.4K
```

```
AVERAGE:             0.5%    20.6%   9       51.0ms  0.0     78      1       31.7K   284.5K

TOTAL:         1.5MB/ 0.6GB           28      153.1ms 0.0     233     4       95.2K   853.5K
```

Momentaufnahme in der Mitte des Tests (bei ca. 1600 Proben):

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE             USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211   0.3%    52.0%   10      1.5ms   0.0     33      0       1750    67.2K
141.62.66.90:11211   0.2%    39.6%   6       15.7ms  0.0     25      0       876     13.7K
141.62.66.93:11211   1.7%    0.0%    8       20.5ms  0.0     25      0       727     1.4M

AVERAGE:             0.7%    30.5%   8       12.6ms  0.0     28      0       1118    488.1K

TOTAL:         2.4MB/ 0.6GB           24      37.7ms  0.0     83      0       3353    1.4M
```

Momentaufnahme nach Abschluss des Tests (nach ca. 3200 Proben):

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE             USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211   0.3%    59.5%   1       0.8ms   0.0     7       0       335     7559
141.62.66.90:11211   0.2%    46.3%   1       1.3ms   0.0     4       0       146     1114
141.62.66.93:11211   1.7%    0.0%    1       1.3ms   0.0     5       0       162     256.8K

AVERAGE:             0.7%    35.2%   1       1.1ms   0.0     5       0       214     88.4K

TOTAL:         2.4MB/ 0.6GB           3       3.4ms   0.0     16      0       643     265.2K
```

Kumulierte Werte am Ende des Tests (nach 3200 Proben):

```
memcache-top v0.6 (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE             USAGE   HIT %   CONN    TIME    EVICT   READ    WRITE
141.62.66.91:11211   0.3%    59.5%   1       0.9ms   0.0     1.4M    13.3M
141.62.66.90:11211   0.2%    46.3%   1       1.3ms   0.0     1.1M    12.0M
141.62.66.93:11211   1.7%    77.1%   1       1.3ms   0.0     1.7M    0.4G

AVERAGE:             0.7%    60.9%   1       1.1ms   0.0     1.4M    0.1G

TOTAL:         2.4MB/ 0.6GB           3       3.4ms   0.0     4.2M    0.4G
```

3.5.2. Test 2: memcached vorgewärmt, kein MySQL Proxy

In diesem Versuch wurden die memcached-Server nicht vor Beginn des Tests neu gestartet, sondern enthielten bereits Cache-Objekte aus vorherigen Tests. Der MySQL Proxy kam in diesem Versuchsteil nicht zum Einsatz.

JMeter-Messergebnisse

	Proben	Durchschnitt	Mittel	90% Marke	Min	Max	% Fehler	Durchsatz	KB/sek
Gesamt	3200	1034 ms	1020 ms	1309 ms	225 ms	12551 ms	0.0	9,05	620,67

Tabelle 6: memcached Test 2 - memcached vorgewärmt, kein MySQL Proxy

Cacti-Graphen

Memcached Server 1 (141.62.66.93)

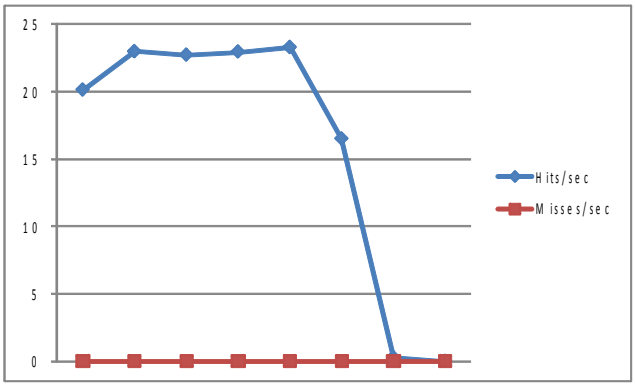

Abbildung 16: memcached 1 - Cache Hits and Misses

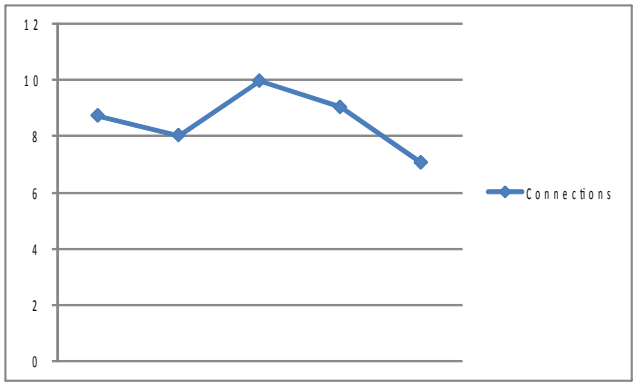

Abbildung 17: memcached 1 - Current Connections

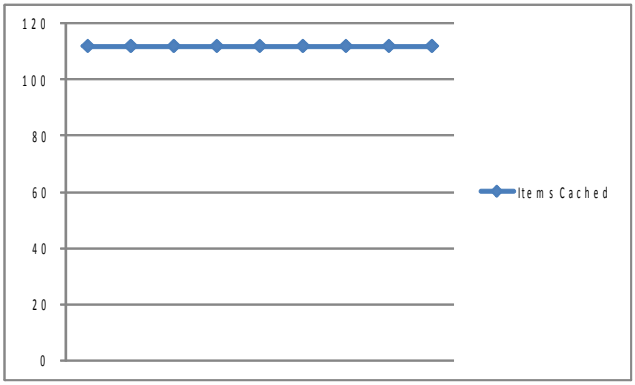

Abbildung 18: memcached 1 - Items Cached

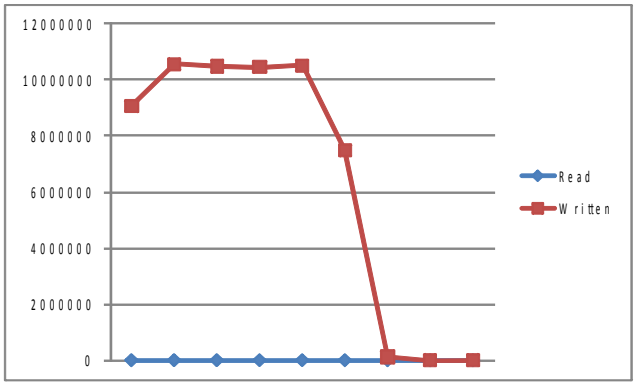

Abbildung 19: memcached 1 - Network Traffic (Bits/sec)

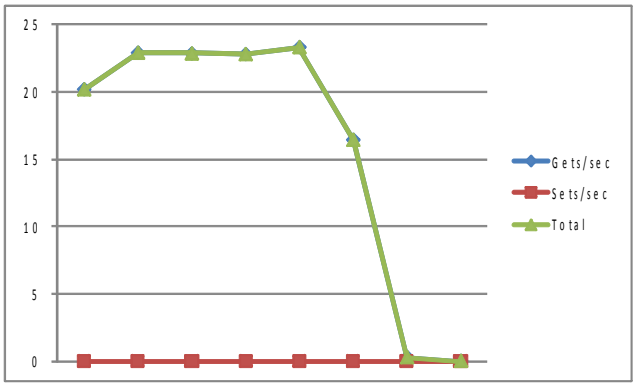

Abbildung 20: memcached 1 - Requests

Memcached Server 2 (141.62.66.90)

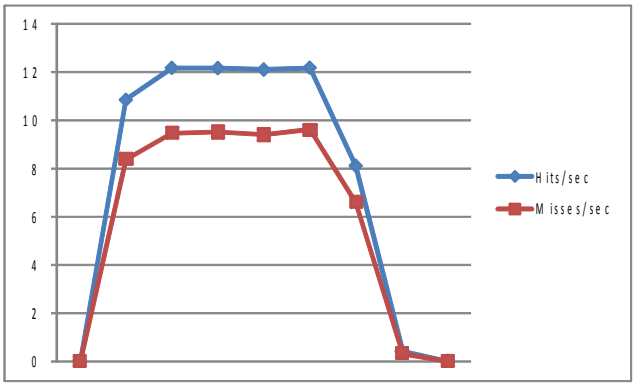

Abbildung 22: memcached 2 - Cache Hits and Misses

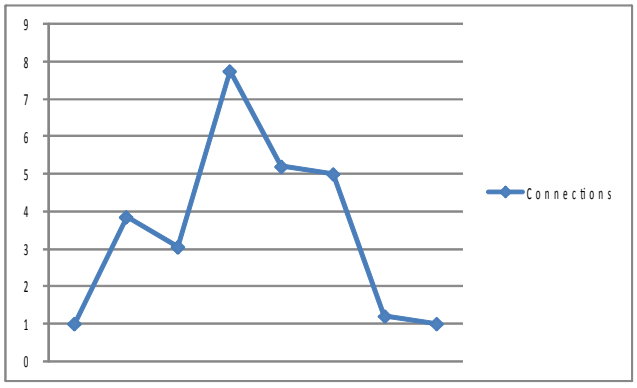

Abbildung 21: memcached 2 - Current Connections

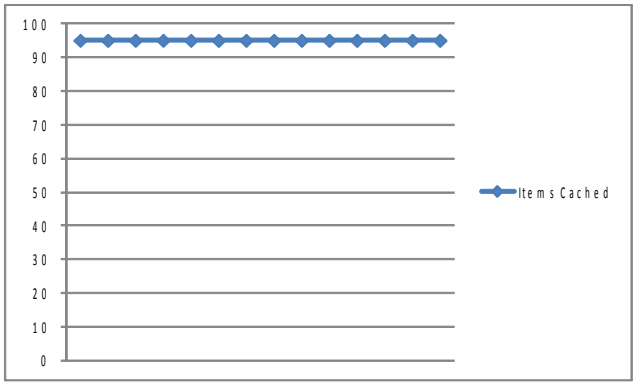

Abbildung 23: memcached 2 - Items Cached

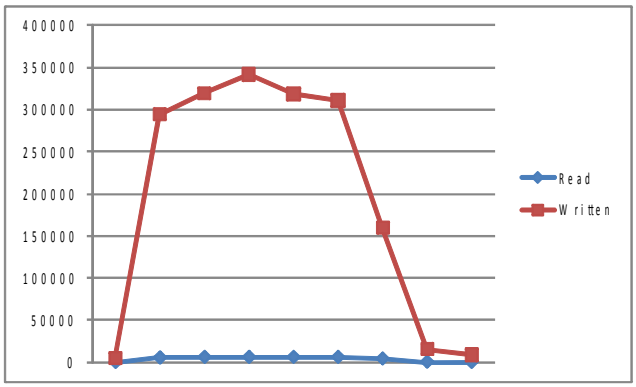

Abbildung 24: memcached 2 - Network Traffic (Bits/sec)

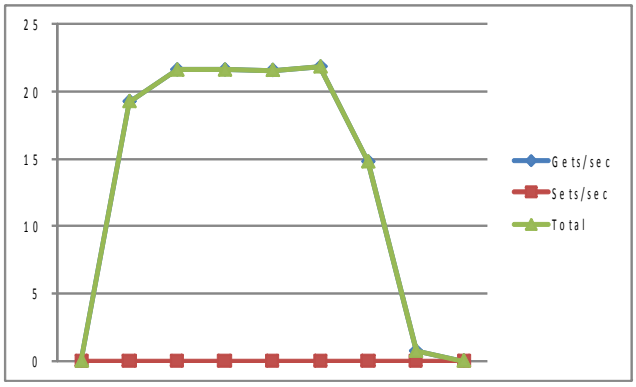

Abbildung 25: memcached 2 - Requests/sec

Memcached Server 3 (141.62.66.91)

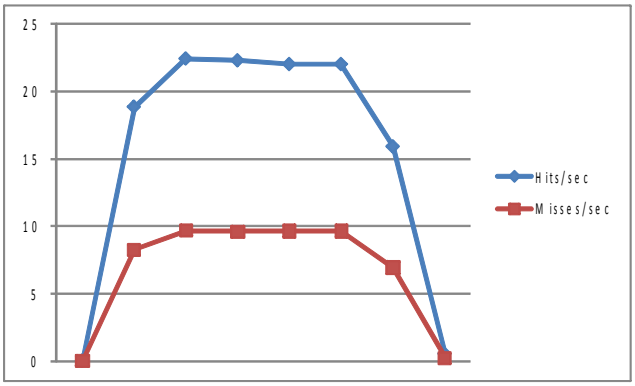

Abbildung 26: memcached 3 - Cache Hits and Misses

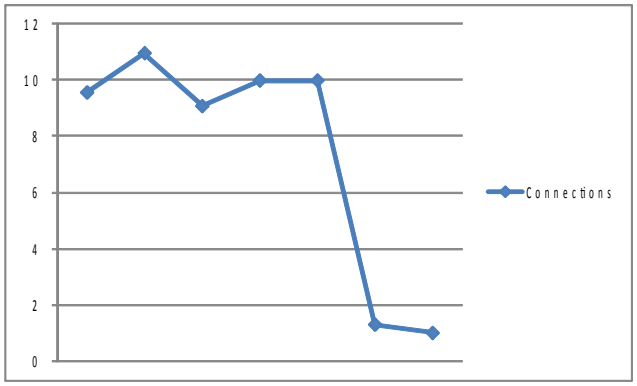

Abbildung 27: memcached 3 - Current Connections

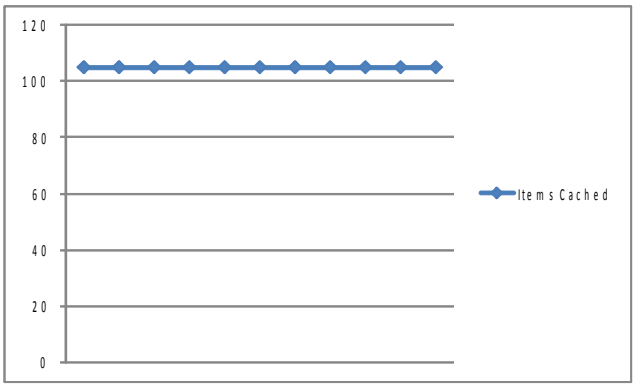

Abbildung 28: memcached 3 - Items Cached

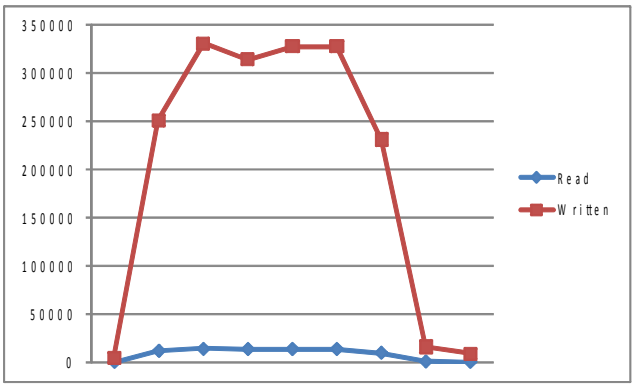

Abbildung 29: memcached 3 - Network Traffic (Bits/sec)

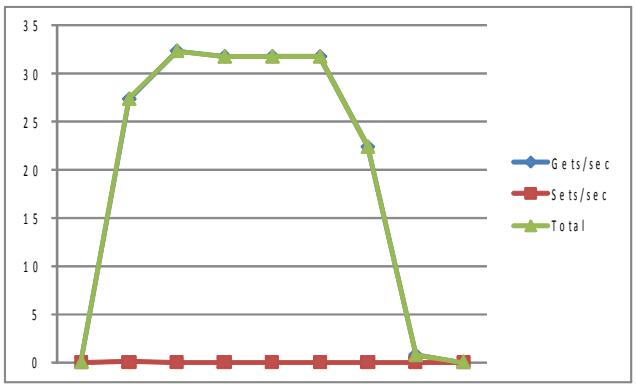

Abbildung 30: memcached 3 - Requests/sec

memcached-top Ausgaben

Momentaufnahme zu Beginn des Tests:

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE              USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211    0.3%    60.0%   11      31.7ms  0.0     34      0       1889    90.3K
141.62.66.90:11211    0.2%    46.8%   3       33.8ms  0.0     23      0       800     59.4K
141.62.66.93:11211    1.7%    0.0%    9       11.7ms  0.0     31      0       901     1.7M
```

```
AVERAGE:             0.7%    35.6%   7       25.7ms  0.0     29      0       1197    627.3K

TOTAL:        2.4MB/ 0.6GB           23      77.2ms  0.0     88      0       3590    1.8M
```

Momentaufnahme in der Mitte des Tests (bei ca. 1600 Proben):

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE             USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211   0.3%    62.5%   11      3.3ms   0.0     36      0       1885    54.0K
141.62.66.90:11211   0.2%    49.1%   8       8.3ms   0.0     21      0       726     1663
141.62.66.93:11211   1.7%    0.0%    10      58.2ms  0.0     26      0       797     1.4M

AVERAGE:             0.7%    37.2%   9       23.3ms  0.0     28      0       1136    497.3K

TOTAL:        2.4MB/ 0.6GB           29      69.8ms  0.0     83      0       3408    1.5M
```

Momentaufnahme am Ende des Tests (bei ca. 3200 Proben):

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE             USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211   0.3%    64.1%   8       30.0ms  0.0     31      0       1753    38.4K
141.62.66.90:11211   0.2%    50.6%   3       1.3ms   0.0     19      0       661     23.9K
141.62.66.93:11211   1.7%    0.0%    3       38.5ms  0.0     20      0       589     1.1M

AVERAGE:             0.7%    38.2%   4       23.3ms  0.0     23      0       1001    387.7K

TOTAL:        2.4MB/ 0.6GB           14      69.9ms  0.0     70      0       3003    1.1M
```

Kumulierte Werte am Ende des Tests bei 3200 Proben:

```
memcache-top v0.6 (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE             USAGE   HIT %   CONN    TIME    EVICT   READ    WRITE
141.62.66.91:11211   0.3%    59.5%   1       0.9ms   0.0     1.4M    13.3M
141.62.66.90:11211   0.2%    46.3%   1       1.3ms   0.0     1.1M    12.0M
141.62.66.93:11211   1.7%    77.1%   1       1.3ms   0.0     1.7M    0.4G

AVERAGE:             0.7%    60.9%   1       1.1ms   0.0     1.4M    0.1G

TOTAL:        2.4MB/ 0.6GB           3       3.4ms   0.0     4.2M    0.4G
```

3.5.3. Test 3: memcached nicht vorgewärmt, mit MySQL Proxy

In diesem Versuchsteil wurden die Caches vor Beginn des Tests geleert und es wurde der in 3.1.3. beschriebene MySQL Proxy eingesetzt, um die Datenbank-Anbindung zu verlangsamen. Da dies zu erheblich längeren Testdauern führte, wurden in diesem Fall weniger Proben genommen als bei den vorherigen Versuchsteilen.

JMeter-Messergebnisse

Information: Der Test wurde nach circa 25 Minuten aufgrund der hohen Dauer abgebrochen.

	Proben	Durchschnitt	Mittel	90% Marke	Min	Max	% Fehler	Durchsatz	KB/sek
Gesamt	529	18902 ms	11369 ms	33231 ms	809 ms	273524 ms	0,01	0,53	37,34

Tabelle 7: memcached Test 3 - memcached nicht vorgewärmt, mit MySQL-Proxy

Cacti-Graphen

Memcached Server 1 (141.62.66.93)

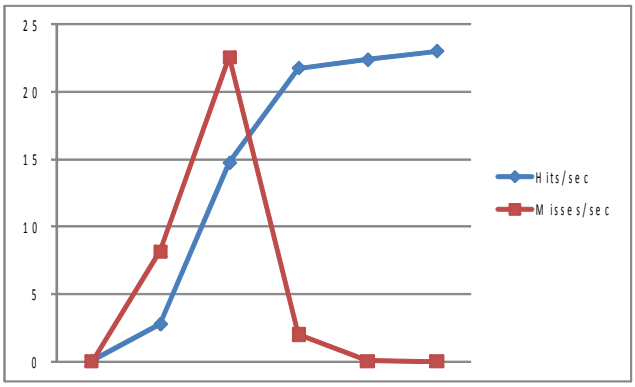

Abbildung 31: memcached 1 - Cache Hits and Misses

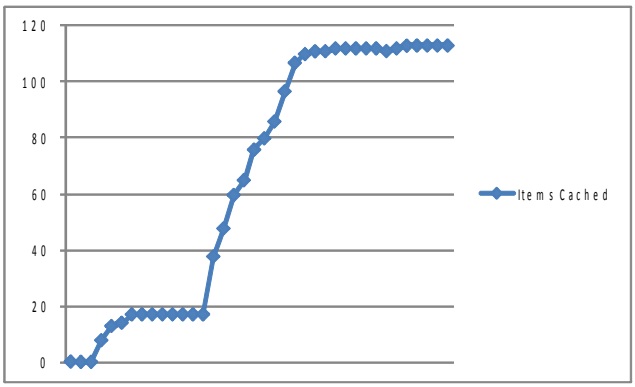

Abbildung 32: memcached 1 - Items Cached

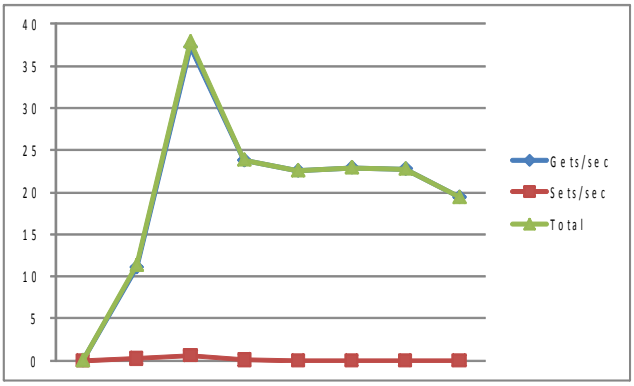

Abbildung 33: memcached 1 - Requests

Memcached Server 2 (141.62.66.90)

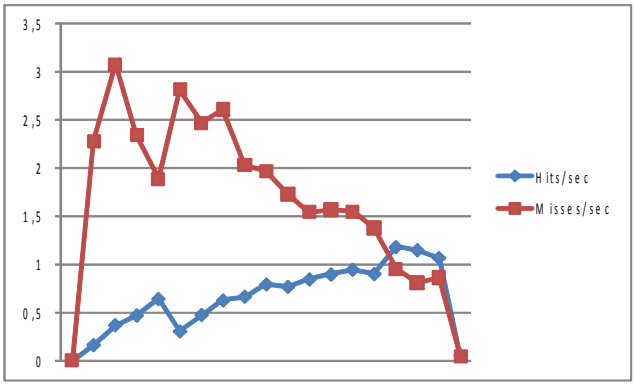

Abbildung 35: memcached 2 - Cache Hits and Misses

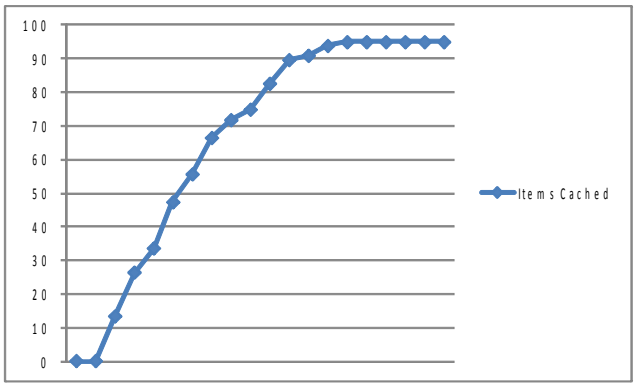

Abbildung 34: memcached 2 - Items Cached

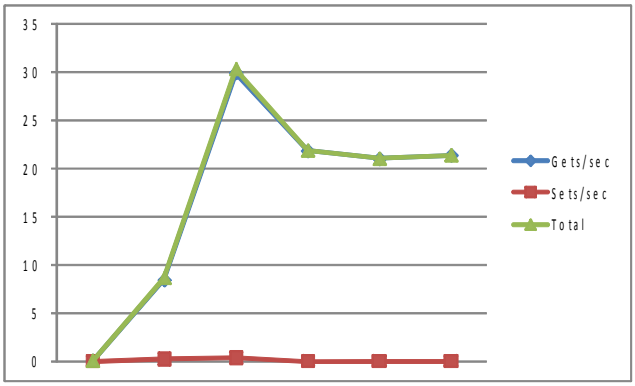

Abbildung 36: memcached 2 - Requests/sec

Memcached Server 3 (141.62.66.91)

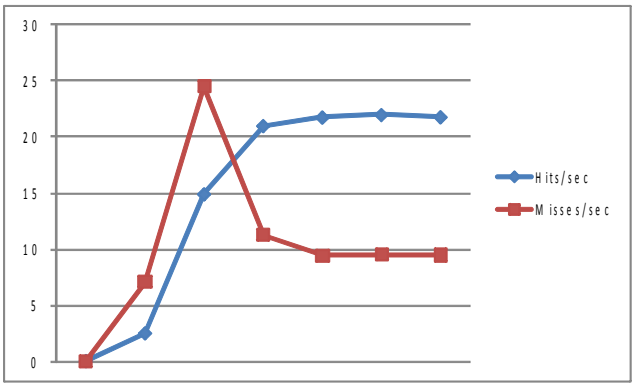

Abbildung 37: memcached 3 - Cache Hits and Misses

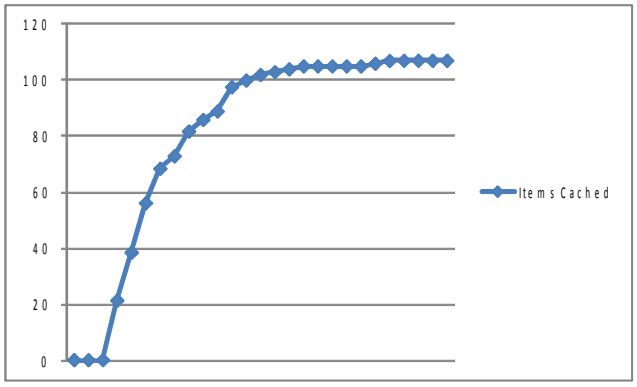

Abbildung 38: memcached 3 - Items Cached

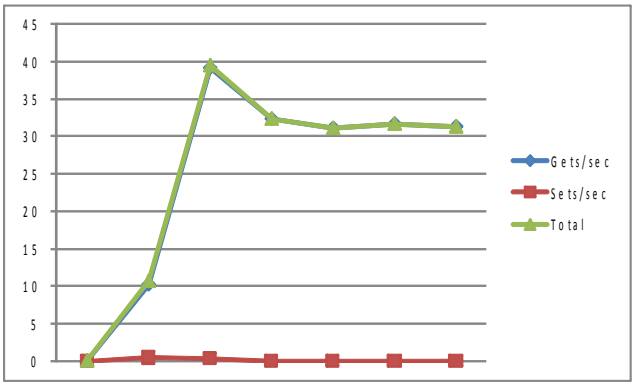

Abbildung 39: memcached 3 - Requests/sec

memcached-top Ausgaben

Momentaufnahme zu Beginn des Tests:

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE              USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211    0.0%    0.0%    11      0.5ms   0.0     1       6       2930    1039
141.62.66.90:11211    0.0%    8.3%    5       2.8ms   0.0     3       1       2865    985
141.62.66.93:11211    0.2%    30.0%   11      1.1ms   0.0     8       1       24.1K   1018

AVERAGE:              0.1%    12.8%   9       1.5ms   0.0     4       3       9.9K    1014

TOTAL:        190.1KB/        0.6GB           27      4.4ms   0.0     12      8       29.8K   3042
```

Momentaufnahme bei ca. 70 Proben:

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE              USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211    0.2%    18.1%   11      0.8ms   0.0     3       0       127     1011
141.62.66.90:11211    0.1%    13.9%   10      1.6ms   0.0     6       0       168     1017
141.62.66.93:11211    1.1%    15.2%   10      1.3ms   0.0     6       0       184     127.0K

AVERAGE:              0.5%    15.7%   10      1.2ms   0.0     5       0       160     43.0K

TOTAL:        1.5MB/  0.6GB           31      3.6ms   0.0     15      0       479     128.9K
```

Momentaufnahme bei ca. 300 Proben:

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE              USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211    0.3%    28.9%   11      0.9ms   0.0     5       0       146     1584
141.62.66.90:11211    0.2%    21.5%   8       1.6ms   0.0     1       0       42      1001
141.62.66.93:11211    1.7%    28.0%   9       1.2ms   0.0     2       0       63      1012

AVERAGE:              0.7%    26.1%   9       1.2ms   0.0     3       0       84      1199

TOTAL:        2.3MB/  0.6GB           28      3.7ms   0.0     8       0       251     3597
```

Momentaufnahme bei ca. 500 Proben:

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE              USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211    0.3%    0.0%    11      0.8ms   0.0     0       0       57      1011
141.62.66.90:11211    0.2%    27.2%   7       1.5ms   0.0     2       0       79      1052
141.62.66.93:11211    1.7%    0.0%    7       1.3ms   0.0     2       0       64      127.0K
```

```
AVERAGE:              0.7%    9.1%    8       1.2ms   0.0     1       0       67      43.0K

TOTAL:        2.4MB/  0.6GB           25      3.7ms   0.0     4       0       200     129.0K
```

Kumulierte Werte des Tests nach 500 Proben:

```
memcache-top v0.6 (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE              USAGE   HIT %   CONN    TIME    EVICT   GETS    SETS    READ    WRITE
141.62.66.91:11211    0.3%    37.2%   1       0.9ms   0.0     3483    171     909.1K  3.1M
141.62.66.90:11211    0.2%    27.8%   1       1.3ms   0.0     2654    99      775.1K  3.2M
141.62.66.93:11211    1.7%    38.9%   1       1.2ms   0.0     3231    134     1.6M    69.5M

AVERAGE:              0.7%    34.7%   1       1.1ms   0.0     3123    135      1.1M   25.3M

TOTAL:        2.4MB/  0.6GB           3       3.4ms   0.0     9368    404      3.3M   75.9M
```

3.5.4. Test 4: memcached vorgewärmt, mit MySQL Proxy

In diesem Versuchsteil wurde die Datenbank durch den MySQL Proxy künstlich verlangsamt. Die memcached-Server wurden vor Beginn der Tests nicht geleert und enthielten zu Testbeginn Cache-Objekte.

JMeter-Messergebnisse

	Proben	Durchschnitt	Mittel	90% Marke	Min	Max	% Fehler	Durchsatz	KB/sek
Gesamt	310	23168 ms	12103 ms	55453 ms	2004 ms	280388 ms	0,03	0,43	26,6

Tabelle 8: memcached Test 4 - memcached vorgewärmt, mit MySQL-Proxy

Cacti-Graphen

Memcached-Server 1 (141.62.66.93)

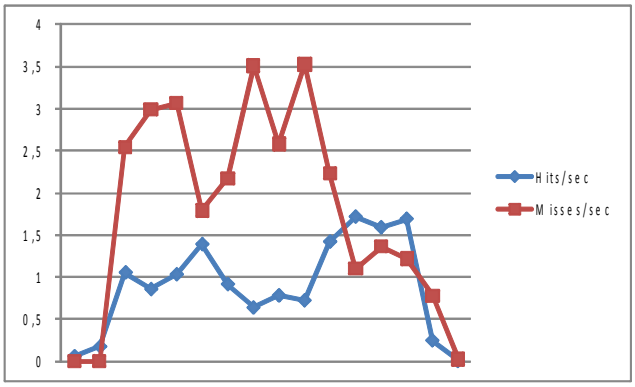

Abbildung 40: memcached 1 - Cache Hits and Misses

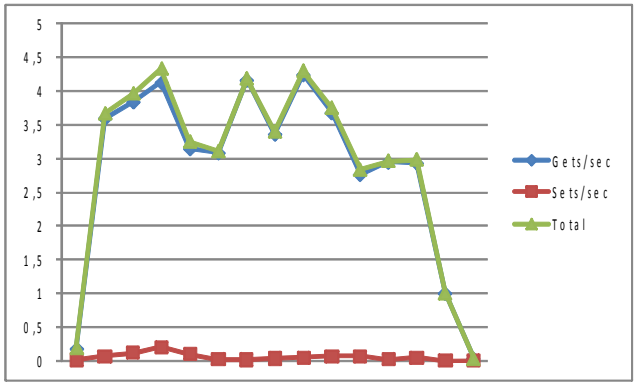

Abbildung 41: memcached 1 - Requests

Memcached-Server 2 (141.62.66.90)

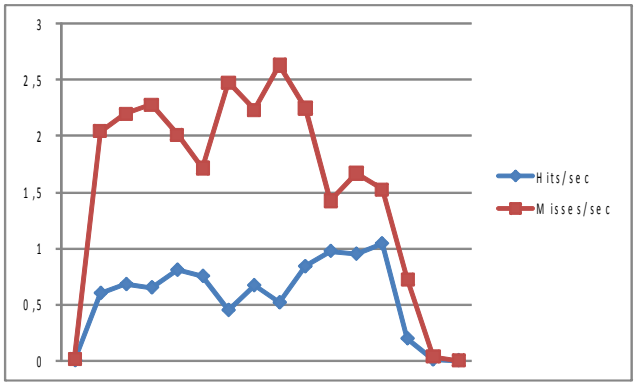

Abbildung 42: memcached 2 - Cache Hits and Misses

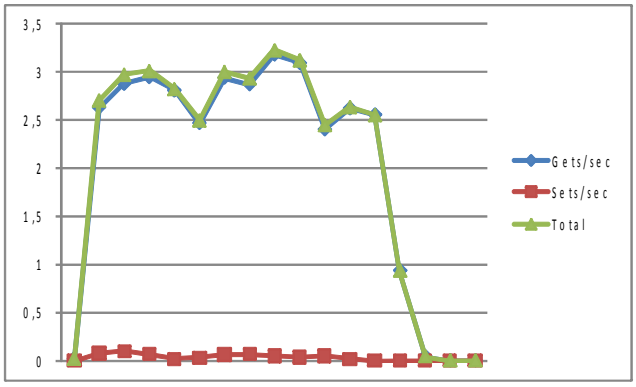

Abbildung 43: memcached 2 - Requests/sec

Memcached-Server 3 (141.62.66.91)

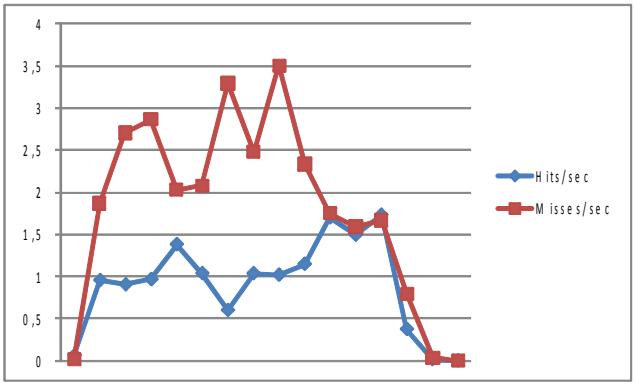

Abbildung 44: memcached 3 - Cache Hits and Misses

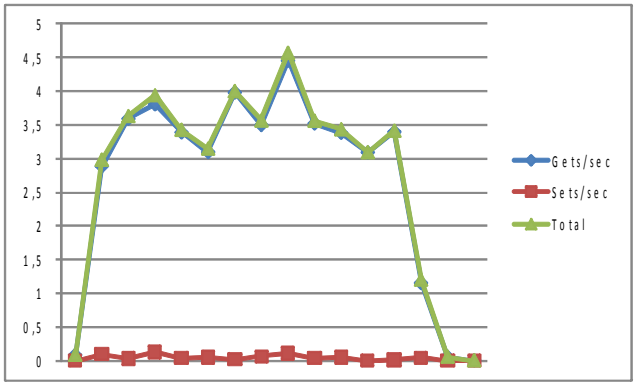

Abbildung 45: memcached 3 - Requests/sec

memcached-top Ausgaben

Momentaufnahme zu Beginn des Tests:

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE               USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211     0.3%    0.0%    11      0.9ms   0.0     2       0       202     9607
141.62.66.90:11211     0.2%    43.5%   10      1.2ms   0.0     6       0       450     13.1K
141.62.66.93:11211     1.6%    69.9%   10      1.2ms   0.0     11      0       406     530.2K

AVERAGE:               0.7%    37.8%   10      1.1ms   0.0     6       0       353     184.2K

TOTAL:        2.3MB/  0.6GB           31      3.4ms   0.0     19      0       1058    552.6K
```

Momentaufnahme bei ca. 150 Proben:

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE              USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211    0.3%    53.0%   11      0.9ms   0.0     5       0       147     1592
141.62.66.90:11211    0.2%    41.3%   8       1.6ms   0.0     3       0       92      1023
141.62.66.93:11211    1.6%    64.0%   9       1.3ms   0.0     0       0       12      1010

AVERAGE:              0.7%    31.4%   9       1.2ms   0.0     3       0       84      1208

TOTAL:        2.3MB/  0.6GB           28      3.7ms   0.0     8       0       251     3625
```

Momentaufnahme bei ca. 300 Proben:

```
memcache-top v0.6     (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE              USAGE   HIT %   CONN    TIME    EVICT/s GETS/s SETS/s  READ/s  WRITE/s
141.62.66.91:11211    0.3%    52.7%   5       0.8ms   0.0     6       0       171     1595
141.62.66.90:11211    0.2%    41.0%   5       1.2ms   0.0     7       0       208     1089
141.62.66.93:11211    1.7%    63.9%   5       1.3ms   0.0     6       0       163     1040

AVERAGE:              0.7%    52.5%   5       1.1ms   0.0     6       0       181     1241

TOTAL:        2.4MB/  0.6GB           15      3.3ms   0.0     19      0       542     3724
```

Kumulierte Werte des Tests nach 500 Proben:

```
memcache-top v0.6 (default port: 11211, color: on, refresh: 1 seconds)

INSTANCE              USAGE   HIT %   CONN    TIME    EVICT   GETS    SETS    READ    WRITE
141.62.66.91:11211    0.3%    52.6%   1       0.7ms   0.0     18.2K   263     2.4M    19.5M
141.62.66.90:11211    0.2%    40.9%   1       1.5ms   0.0     13.2K   173     1.9M    18.0M
141.62.66.93:11211    1.7%    63.8%   1       1.2ms   0.0     15.3K   243     3.3M    0.5G

AVERAGE:              0.7%    52.4%   1       1.2ms   0.0     15.6K   226     2.5M    0.2G

TOTAL:        2.4MB/ 0.6GB            3       3.5ms   0.0     46.7K   679     7.6M    0.6G
```

3.6 Erkenntnisse aus den memcached-Versuchen

Im Rahmen der Vorlesung wurde memcached im Umfeld eines Wiki (Wikimedia) untersucht. Dazu wurden mehrere Tests mit jeweils anderen Konfigurationen durchgeführt, um Caching-Effekte durch memcached zu provozieren. Die in den vorigen Kapiteln beschriebenen Tests wurden mit JMeter durchgeführt. Die Ergebnisse dieser JMeter Messungen zeigt folgende Tabelle:

Tests	Proben	Durchschnitt (in ms)	Mittel (in ms)	90% Marke (in ms)	Min (in ms)	Max (in ms)	% F.	Durchsatz	KB/sek
Referenz Tests									
R-1: kein memcached, kein mysql-proxy	3200	1095	911	1290	228	117264	0	8,32	531,26
R-2: kein memcached, mysql-proxy	510	23424	15789	42209	1955	265339	0,02	0,43	27,16
Memcached Tests									
Test-1 memcached leer	3200	1103	901	1276	223	134099	0	8,01	524,72
Test-2 memcached vorgewärmt	3200	1034	1020	1309	225	12551	0	9,05	620,67
Test-3 memcached, mysql-proxy	529	18902	11369	33231	809	273524	0,01	0,53	37,34
Test-4 memcached vorgewärmt, mysql-proxy	310	23168	12103	55453	2004	280388	0,03	0,43	26,6

Tabelle 9: memcached - Vergleich der Messungen

Alle Tests werden in Referenz zu einem Test ohne jegliche Art von Caching gesehen. Dabei unterscheidet sich jedoch je nach Szenario ob der MySQL Proxy an ist oder nicht. Die Ergebnisse dieses Tests sind ebenfalls der Tabelle (siehe Referenz-Tests) zu entnehmen.

Bei den folgenden Tests wurde zuerst ein frisch gestarteter memcached getestet, die Ergebnisse wurden dann mit einem zweiten Test mit vorgewärmten memcached durchgeführt, um die Auswirkungen eines vorgefüllten Caches zu verdeutlichen. Die Konfiguration umfasste wie bei allen weiteren Tests drei Server. Die Datenbank wurde vom Server, auf dem der HTTP-Dienst Apache lief, auf einen anderen Server umgezogen, um die Auswirkungen von memcached zu verstärken und messbar zu machen.

Vergleicht man die Werte zwischen den ersten beiden memcached Tests und dem Referenztest R-1 fällt auf, dass sich die Werte im Durchschnitt und im Mittel nicht signifikant zu den Werten mit der Benutzung von memcached verändert haben. Die Schlussfolgerung daraus ist, dass ein lokaler Datenbank-Zugriff vermutlich in diesem Szenario schneller beziehungsweise gleich schnell ist, wie der Zugriff unter Verwendung von memcached. Ein Effekt von memcached ist allerdings am Durchsatz und an der Datenrate zu erkennen, die beim vorgewärmten memcached etwas höher lagen. Vergleicht man den Unterschied zwischen einem anfangs leeren Cache und einem vorgewärmten Cache, zeigt sich ein leichter Effekt im Durchschnitt, in der Datenrate und dem Daten-Durchsatz.

Um besser nachvollziehen zu können, ob ein Datenbank-Zugriff schneller/gleich schnell ist wie ein memcached-Zugriff und dadurch den Effekt von memcached besser messen zu können, wurde die Datenbank durch einen MySQL Proxy künstlich ausgebremst. Die Auswirkung dessen war, dass ein Datenbank-Zugriff rund 300 Millisekunden dauerte, der vorher im nicht messbaren Bereich lag. Vergleicht man die Werte, lässt sich deutlich der Effekt der MySQL-Bremse erkennen. Der Werte fallen in jedem Bereich extrem ab. Requests dauern im Schnitt 18 Sekunden und die Datenrate sinkt auf 37 KB/s. Im Vergleich mit dem 2. Referenztest zeigt sich nun deutlicher, dass sich durch den Einsatz von memcached die Performance verbessert. Auffallend ist jedoch, dass sich bei vorgewärmten Cache die Werte nicht – wie man vermuten könnte – verbessern. Dies könnte durch die geringe Anzahl von Proben bedingt sein.

Wie im Einführungskapitel zu memcached beschrieben puffert memcached – im Gegensatz zu Squid (siehe folgendes Kapitel) – nicht ganze HTML-Seiten der Wiki-Artikel, sondern nur gewisse Datenbankobjekte. Da trotz memcached noch Daten aus der Datenbank abgefragt werden müssen, kann memcached den verzögernden Effekt des MySQL Proxys nicht egalisieren.

4. Squid

Um den Aspekt des Cachings neben dem verteilten Objekt-Caching auf einer weiteren Ebene der Anwendungsarchitektur zu betrachten, wurde im Rahmen des Seminars die Technologie Squid untersucht.

4.1 Einführung

Squid ist ein freier, quell-offener, cachender Webproxy. Er dient dazu, den HTTP-Anfragen schneller und effizienter zu beantworten. URIs[33], die in einem Netzwerk oft abgerufen werden, werden von Squid im Cache vorgehalten und – sofern sie noch aktuell sind – direkt an den Client ausgeliefert. Neben Protokollen wie HTTP und HTTPS werden weitere Protokolle wie FTP und das Caching von Datenbankanfragen unterstützt.

Squid optimiert den Datenfluss zwischen Client und Server, erhöht die Performance und ermöglicht es, höhere Zugriffszahlen zu bedienen. Neben dem einfachen Puffern und Ausliefern von Inhalten, kann Squid Requests an Server delegieren, die beispielsweise auch einen Squid Webproxy bereitstellen. Auf diese Weise kann eine Cache-Server-Hierarchie aufgebaut werden, um die Effizienz großer Netzwerke (wie beispielsweise die von ISPs[34]) zu steigern.

Squid wird jedoch nicht nur als Proxy für Netzwerke und ISPs eingesetzt. Es wird ebenfalls als Cache für die Inhalts-Auslieferung (Content-Delivery) von Webseiten eingesetzt um die Auslieferung des Inhalts an einen Client zu beschleunigen. Es kann, über mehrere Server verteilt, ein Content-Cluster aufgebaut werden, bei dem die Auslieferung je nach Standort von einem anderen Server vorgenommen wird.

Zum erfolgreichen Einsatz von Squid für MediaWiki im Falle von Wikipedia macht die Wikimedia Foundation folgende Aussage:

”The Squid systems are currently running at a hit-rate of approximately 75%, effectively quadrupling the capacity of the Apache servers behind them. This is particularly noticeable when a large surge of traffic arrives directed to a particular page via a web link from another site, as the caching efficiency for that page will be nearly 100%.“[35]

33 Uniform Resource Identifiers

34 Internet Service Providers

35 vgl. http://www.squid-cache.org/

4.2 Konfiguration

Zur Einrichtung von Squid für die im Seminar verwendete MediaWiki-Installation wurden die in den folgenden Abschnitten aufgeführten Konfigurationen gemäß der Anleitung im Wiki von MediaWiki[36] durchgeführt.

4.2.1. Squid-Konfiguration

Squid wurde als RPM-Paket auf dem Server 141.62.66.91 installiert und wie folgt konfiguriert:

```
/etc/squid/squid.conf
```

```
http_port 141.62.66.91:80 defaultsite=141.62.66.91 vhost
cache_peer 127.0.0.1 parent 80 0 no-query originserver round-robin name=wiki
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl web_ports port 80
http_access allow web_ports
http_access allow manager localhost
http_access deny manager
acl purge method PURGE
http_access allow purge localhost
http_access deny purge
http_access deny all
hierarchy_stoplist cgi-bin ?
cache_mem 512 MB
maximum_object_size_in_memory 4096 KB
cache_dir ufs /var/spool/squid 20000 16 256
access_log /var/log/squid/access.log squid
cache_log /var/log/squid/cache.log
cache_store_log /var/log/squid/store.log
acl QUERY urlpath_regex cgi-bin \?
refresh_pattern ^ftp:           1440    20%     10080
refresh_pattern ^gopher:        1440    0%      1440
refresh_pattern .               0       80%     4320
acl apache rep_header Server ^Apache
broken_vary_encoding allow apache
forwarded_for on
coredump_dir /var/spool/squid
```

Die aufgeführten Squid-Konfigurationsoptionen werden im einzelnen in der Squid-Dokumentation näher erläutert[37].

36 vgl. http://www.mediawiki.org/wiki/Manual:Squid_caching

37 vgl. http://www.squid-cache.org/Doc/config/

Relevant sind vor allem folgende Einstellungen:

- Squid „übernimmt“ auf dem Server 141.62.66.91 Port 80, um Requests entgegenzunehmen (`http_port`). Sein Cache-Peer ist der Apache-Server auf dem gleichen Netzwerkknoten, der jedoch nur noch über das Loopback-Interface auf Port 80 hört und dort Requests von Squid entgegennehmen kann (`cache_peer`).
- Squid erhält 512 KB Arbeitsspeicher (`cache_mem`) und 20.000MB Festplattenspeicher (`cache_dir`).
- Die Größe der im RAM speicherbaren einzelnen Objekte wird auf 4MB gesetzt (`maximum_object_size_in_memory`).
- Cache-Inhalte ohne Ablaufzeit werden für maximal 4320min als noch gültig angenommen. Bis zu 80% dieses Alters werden sie auf jeden Fall als noch aktuell angenommen (`refresh_pattern .`).

4.2.2. httpd-Konfiguration

Der Apache http-Daemon wird die Bind-Adresse des Loopback-Interfaces konfiguriert:

```
/etc/httpd/conf/httpd.conf
```

```
Listen 127.0.0.1:80
ServerName 127.0.0.1:80
```

4.2.3. Mediawiki-Konfiguration

Für MediaWiki werden die direkt unterstützten Squid-Optionen konfiguriert:

```
/var/www/html/wiki/LocalSettings.php
```

```
# SQUID
$wgUseSquid = true;
$wgSquidServers = array('127.0.0.1');
$wgSquidMaxage = 2678400;
$wgCachePages = true;
$wgUseETag = true;
$wgUseESI = false;
```

Diese Konfigurationsoptionen werden detailliert im Wiki des MediaWiki-Projektes erläutert[38].

38 vgl. http://www.mediawiki.org/wiki/Manual:Configuration_settings

5. Squid-Versuche

5.1 Versuchsbeschreibung

Für die Squid-Versuche wurde wie ein JMeter-Test verwendet, der in jedem Durchlauf aus einer Auswahl von 93 Wikipedia-Artikeln zufällig 10 abfragt (lesend). Die Messung wurde mit 5 parallelen Threads mit einer Ramp-Up-Zeit von 3s sowie einer Wiederhol-Rate von 20 durchgeführt.

Die Messung wurde wie im Falle der memcached-Versuche von einem Rechner im Labor durchgeführt, der über Ethernet-Kabel direkt an den 100BaseT-Switch der Server angeschlossen war.

Zur Vermeidung von Cache-Interferenzen wurde sowohl memcached als auch der MySQL-Query Cache für die Messungen deaktiviert.

Wie oben bereits erwähnt, wurde Squid auf dem gleichen Hardware-Server wie Apache betrieben (141.62.66.91). Der Datenbank-Server wurde wie im Falle der memcached-Messungen auf einen anderen Server umgezogen (141.62.66.90). Für Test 3 (s.u.) wurde der in 3.1.3. beschriebene MySQL Proxy eingesetzt.

5.2 Test 1: ohne Squid-Cache, ohne MySQL Proxy

Im ersten Versuchtsteil wurde eien Referenzmessung für den JMeter Test ohne aktivierten Squid sowie ohne MySQL-Proxy durchgeführt.

JMeter-Messergebnisse

Test	Proben	Durchschnitt	Mittel	90% Marke	Min	Max	% F.	Durchsatz	KB/sek
Kein squid, ungedrosselt, DB lokal	1000	662 ms	505 ms	703 ms	227 ms	37747 ms	0	6,97	454,94

Tabelle 10: Squid Test1 - ohne Squid-Cache, ohne MySQL-Proxy

5.3 Test 2: mit vorgewärmtem Squid-Cache, ohne MySQL Proxy

Im zweiten Versuchsteil wurde Squid aktiviert und vor Beginn des Versuchs nicht geleert. Der MySQL Proxy war in diesem Test deaktiviert.

JMeter-Messergebnisse

Test	Proben	Durchschnitt	Mittel	90% Marke	Min	Max	% F.	Durchsatz	KB/sek
Squid, kein mysql-Proxy	1000	31 ms	19 ms	74 ms	2 ms	327 ms	0	113,97	11071,8

Tabelle 11: Squid Test 2- mit vorgewärmtem Squid, ohne MySQL-Proxy

5.4 Test 3: mit vorgewärmtem Squid-Cache, mit MySQL Proxy

In diesem Versuchsteil war Squid aktiviert, vorgewärmt und es wurde der MySQL Proxy eingesetzt, um Datenbankzugriffe zu verlangsamen.

JMeter-Messergebnisse

Test	Proben	Durchschnitt	Mittel	90% Marke	Min	Max	% F.	Durchsatz	KB/sek
Squid, mysql-proxy	1000	29 ms	17 ms	65 ms	2 ms	650 ms	0	121,41	10854,36

Tabelle 12: Squid Test 3 - vorgewärmter Squid-Cache, mit MySQL-Proxy

5.5 Log-Auszüge

5.5.1. HTTP-Header bei Cache-Miss und Hit

HTTP-Header bei Cache-Miss:

```
HTTP/1.0 301 Moved Permanently
Date: Tue, 16 Feb 2010 11:27:05 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Vary: Accept-Encoding,Cookie
X-Vary-Options: Accept-Encoding;list-contains=gzip,Cookie;string-
contains=wikidbToken;string-contains=wikidbLoggedOut;string-contains=wikidb_session
Cache-Control: s-maxage=1200, must-revalidate, max-age=0
Last-Modified: Tue, 16 Feb 2010 11:27:05 GMT
Location: http://141.62.66.91/wiki/index.php/Main_Page
Content-Type: text/html; charset=utf-8
X-Cache: MISS from rn91.rnlabor.hdm-stuttgart.de
X-Cache-Lookup: MISS from rn91.rnlabor.hdm-stuttgart.de:80
Via: 1.0 rn91.rnlabor.hdm-stuttgart.de:80 (squid/2.6.STABLE21)
Connection: close
```

HTTP-Header bei Cache-Hit:

```
HTTP/1.0 200 OK
Date: Tue, 16 Feb 2010 11:20:17 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Content-Language: en
ETag: W/"wikidb:pcache:idhash:312505-0!1!0!!en!2--20090925042123"
Vary: Accept-Encoding,Cookie
X-Vary-Options: Accept-Encoding;list-contains=gzip,Cookie;string-
contains=wikidbToken;string-contains=wikidbLoggedOut;string-contains=wikidb_session
Cache-Control: s-maxage=2678400, must-revalidate, max-age=0
Last-Modified: Tue, 16 Feb 2010 11:16:36 GMT
Content-Type: text/html; charset=UTF-8
Age: 310
Content-Length: 102091
X-Cache: HIT from rn91.rnlabor.hdm-stuttgart.de
X-Cache-Lookup: HIT from rn91.rnlabor.hdm-stuttgart.de:80
Via: 1.0 rn91.rnlabor.hdm-stuttgart.de:80 (squid/2.6.STABLE21)
Connection: close
```

5.6 Log-Auszug

Auszug aus dem Squid-Access-Log (`/var/log/squid/access.log`):

```
1266319935.033    113 141.62.66.88 TCP_HIT/200 194193 GET http://141.62.66.91/wiki/index.-
php/Liste_der_Staatsoberh%C3%A4upter_1215 - NONE/- text/html
1266319935.039     11 141.62.66.88 TCP_HIT/200 24300 GET
http://141.62.66.91/wiki/index.php/Bierkit - NONE/- text/html
1266319935.047     20 141.62.66.88 TCP_MEM_HIT/200 16506 GET
http://141.62.66.91/wiki/index.php/Goldkrone_(M%C3%BCnze) - NONE/- text/html
1266319935.047      1 141.62.66.88 TCP_MEM_HIT/200 17165 GET
http://141.62.66.91/wiki/index.php/Naudhiz - NONE/- text/html
1266319935.047      7 141.62.66.88 TCP_HIT/200 86681 GET
http://141.62.66.91/wiki/index.php/Biogenetische_Grundregel - NONE/- text/html
1266319935.057      1 141.62.66.88 TCP_MEM_HIT/200 22566 GET
http://141.62.66.91/wiki/index.php/R._Madhavan - NONE/- text/html
1266319935.057      0 141.62.66.88 TCP_HIT/301 2278 GET
http://141.62.66.91/wiki/index.php/Julius_Jäger - NONE/- text/html
1266319935.059      1 141.62.66.88 TCP_HIT/301 2278 GET
http://141.62.66.91/wiki/index.php/Julius_Jäger - NONE/- text/html
1266319935.059      0 141.62.66.88 TCP_MEM_HIT/200 22566 GET
http://141.62.66.91/wiki/index.php/R._Madhavan - NONE/- text/html
1266319935.062      0 141.62.66.88 TCP_HIT/200 18856 GET
http://141.62.66.91/wiki/index.php/Julius_J%C3%A4ger - NONE/- text/html
1266319935.065     18 141.62.66.88 TCP_HIT/200 18856 GET
http://141.62.66.91/wiki/index.php/Julius_J%C3%A4ger - NONE/- text/html
1266319935.086     19 141.62.66.88 TCP_MEM_HIT/200 15144 GET
http://141.62.66.91/wiki/index.php/RTF - NONE/- text/html
1266319935.089     14 141.62.66.88 TCP_MEM_HIT/200 22566 GET
http://141.62.66.91/wiki/index.php/R._Madhavan - NONE/- text/html
1266319935.097     35 141.62.66.88 TCP_HIT/200 246567 GET http://141.62.66.91/wiki/index.-
php/The_Reichenberg_Fellowship - NONE/- text/html
1266319935.105     87 141.62.66.88 TCP_HIT/200 194193 GET http://141.62.66.91/wiki/index.-
php/Liste_der_Staatsoberh%C3%A4upter_1215 - NONE/- text/html
1266319935.106     38 141.62.66.88 TCP_HIT/200 18856 GET
http://141.62.66.91/wiki/index.php/Julius_J%C3%A4ger - NONE/- text/html
1266319935.108     40 141.62.66.88 TCP_HIT/200 246567 GET http://141.62.66.91/wiki/index.-
php/The_Reichenberg_Fellowship - NONE/- text/html
1266319935.117      0 141.62.66.88 TCP_HIT/301 2318 GET
http://141.62.66.91/wiki/index.php/Liste_der_Staatsoberhäupter_1215 - NONE/- text/html
1266319935.117      0 141.62.66.88 TCP_HIT/301 2318 GET
http://141.62.66.91/wiki/index.php/Liste_der_Staatsoberhäupter_1215 - NONE/- text/html
1266319935.128      4 141.62.66.88 TCP_HIT/200 24300 GET
http://141.62.66.91/wiki/index.php/Bierkit - NONE/- text/html
1266319935.128      9 141.62.66.88 TCP_MEM_HIT/200 15144 GET
http://141.62.66.91/wiki/index.php/RTF - NONE/- text/html
1266319935.138      0 141.62.66.88 TCP_HIT/301 2318 GET
http://141.62.66.91/wiki/index.php/Liste_der_Staatsoberhäupter_1215 - NONE/- text/html
1266319935.147     19 141.62.66.88 TCP_HIT/200 194193 GET http://141.62.66.91/wiki/index.-
php/Liste_der_Staatsoberh%C3%A4upter_1215 - NONE/- text/html
```

5.7 Erkenntnisse aus den Squid-Versuchen

In unserem MediaWiki Test-Setup ließen sich durch den Einsatz von Squid beträchtliche Verbesserungen der Antwortzeit unserer MediaWiki-Infrastruktur erzielen. Die Auslieferung von Anfragen aus dem Cache ist, verglichen mit direkter Generierung und Auslieferung, um ein Vielfaches effizienter.

Der Testaufbau mit entfernter, nicht über den MySQL Proxy geschleifter Datenbank und ohne den Einsatz von memcached oder Squid, führt zur Auslieferung von rund 530 Kilobytes pro Sekunde. Durch den Einsatz von Squid lässt sich eine rund 20-fache Steigerung erzielen, so dass bei ansonsten vergleichbarem Setup rund 11000 Kilobytes ausgeliefert werden können.

Von besonderem Interesse ist es, den Einfluss der Datenbank-Drosselung mit Hilfe des MySQL Proxy zu betrachten. Ohne den Einsatz von Caching sinkt der Durchsatz mit verlangsamter Datenbank auf etwa 27 Kilobytes pro Sekunden, was rund 5% des ursprünglichen Wertes entspricht. Wird in diesem Setup der Squid-Cache hinzugefügt, kann diese Verschlechterung nahezu vollständig aufgefangen werden. Mit circa 10800 Kilobytes pro Sekunde liegt der erzielte Durchsatz nahezu gleich auf mit dem Wert, der bei ungedrosselter Datenbank erzielt werden kann.

Ein weiterer deutlich sichtbarer Effekt, der durch den Einsatz von Squid erzielt werden kann, ist die massive Verkürzung der Request-Zeiten. Während die Erstellung und Auslieferung einer Seite ohne den Einsatz von Caching durchschnittlich 1095 Millisekunden dauert, kann diese Zeitspanne durch Aktivierung von Squid auf 31 Millisekunden verringert werden. Auch hier zeigt sich wieder, wie effizient die Verlangsamung der Datenbank abgefangen wird: Ohne Caching werden bei aktiviertem MySQL Proxy Requests durchschnittlich in rund 23000 Millisekunden verarbeitet. Wird der befüllte Squid-Cache hinzugenommen liegen die erreichten Zeiten wieder bei rund 30 Millisekunden.

Insgesamt kann somit der Schluss gezogen werden, dass durch den Einsatz von Squid immense Performance-Steigerungen ermöglicht werden. Zusätzlich wird in den Versuchen deutlich, dass eine drastische Reduktion der Datenbank-Zugriffe erreicht wird, was zu einer signifikanten Entlastung des MySQL-Servers führt.

6. Fazit

Die nachfolgend abgedruckte Tabelle gibt nochmals einen Gesamtüberblick der durchgeführten Tests. Die wichtigsten Ergebnisse werden daraus ersichtlich.

Tests	Proben	Durchschnitt (in ms)	Mittel (in ms)	90% Marke (in ms)	Min (in ms)	Max (in ms)	% F.	Durchsatz	KB/sek
Referenz Tests									
R-1: kein memcached, kein mysql-proxy	3200	1095	911	1290	228	117264	0	8,32	531,26
R-2: kein memcached, mysql-proxy	510	23424	15789	42209	1955	265339	0,02	0,43	27,16
R-3: kein squid, kein mysql-Proxy, lokale Datenbank	1000	662	505	703	227	37747	0	6,97	454,94
Memcached Tests									
Test-1 memcached leer	3200	1103	901	1276	223	134099	0	8,01	524,72
Test-2 memcached vorgewärmt	3200	1034	1020	1309	225	12551	0	9,05	620,67
Test-3 memcached, mysql-proxy	529	18902	11369	33231	809	273524	0,01	0,53	37,34
Test-4 memcached vorgewärmt, mysql-proxy	310	23168	12103	55453	2004	280388	0,03	0,43	26,60
Squid Tests									
Squid, kein mysql-Proxy	1000	31	19	74	2	327	0	113,97	11071,80
Squid, mysql-proxy	1000	29	17	65	2	650	0	121,41	10854,36

Tabelle 13: memcached und Squid - Gegenüberstellung aller Tests

Während durch den Einsatz von Squid immense Performance-Steigerungen beobachtet werden konnten, sind die Auswirkungen von memcached in unserer Testumgebung nicht so deutlich, wie zu Beginn erwartet, und im Besonderen nicht so groß, wie auf den zitierten Internetseiten angegeben. Die Gründe hierfür sind zum Einen in unserem spezifischen Testaufbau und der nur begrenzt erzeugbaren Last zu suchen, zum anderen auch in der Art der Inhalte, die im Cache abgelegt werden.

Während durch den Squid-Cache fertige HTML-Seiten im Zwischenspeicher abgelegt werden, verwendet WikiMedia memcached lediglich für das Caching einiger Datenbankobjekte. Tritt also bei Squid ein Cache-Hit auf, wird der gesamte Request direkt aus dem Cache beantwortet. Auf keinem der Backend-Systeme wird somit eine Last erzeugt, was wiederum die Tatsache erklärt, dass die Aktivierung des MySQL Proxy keinen wesentlichen Einfluss auf die Performance hat, sofern der Squid Cache eingesetzt wird. Da memcached hingegen lediglich Metadaten bereithält, werden hier trotz aktiviertem Cache noch vergleichsweise häufig Zugriffe auf die Datenbank durchgeführt. Das Caching kann somit auch nur begrenzte Auswirkungen zeigen.

Da sich mit den zur Verfügung stehenden Mitteln keine Testumgebung realisieren lässt, die auch nur annähernd den Dimensionen von Wikipedia entspricht, sind auch die erzielbaren Effekte nicht so deutlich, wie in den angegebenen Quellen beschrieben. Es ist somit davon auszugehen, dass der bei Caching der Metadaten durch memcached erzielbare Effekt erst ab einer gewissen Skalierungsgröße der Gesamtarchitektur in dem beschriebenen Maße ins Gewicht fällt.