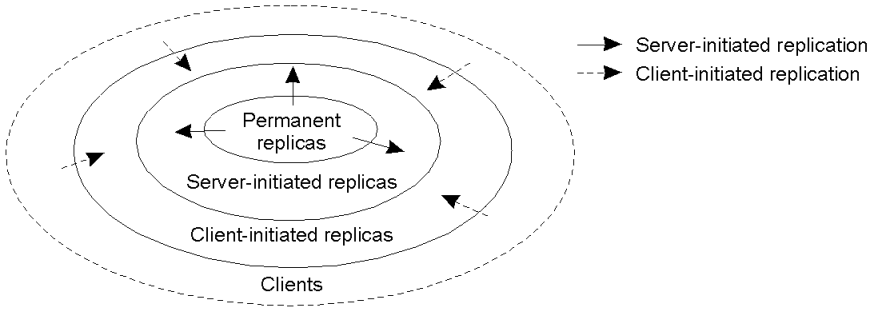
Consistency and Replication

Buch: 7.1, 7.4, 7.5, 10.6, 4.5.2

Fragen:

1. **Was sind die Hauptgründe für den Einsatz von Replikation in verteilten Systemen? In welcher Beziehung stehen Replikation und Skalierbarkeit zueinander? Erläutern Sie in diesem Zusammenhang verschiedene Varianten der Content Replication und des Content Placement.**

Die Hauptgründe für den Einsatz von Replikation sind *Zuverlässigkeit* und *Leistung*. Wurde eine bestimmte Datei repliziert, ist es möglich, durch Wechseln auf eine Kopie die Arbeit fortzusetzen, wenn eines der Replikate abstürzt. Weiters bietet Replikation auch eine Möglichkeit, korrupte Dateien, wie sie etwa im Fall eines fehlerhaften Schreibzugriffes entstehten, zu erkennen. In Fällen, wo ein Server eine Vielzahl von Anfragen zu selben Zeit erfüllen muss, kann die Leistung gesteigert werden, indem der Server repliziert und die Arbeit in der Folge aufgeteilt wird (*Skalierung nach Größe*). Auch bei *geographischer Skalierung* wird auf Replikation zurück gegriffen. Grundidee ist, die Daten in die Nähe jener Prozesse zu platzieren, von denen sie benötigt werden, um so die Zugriffszeit zu verringern. Der *Nachteil* der Replikation ist jedoch, dass Replikate Mehraufwand bedeuten, da die Daten konsistent gehalten werden müssen, was sich insbesondere in jenen Fällen negativ auswirkt, wo die Zahl der Updatevorgänge die Zahl der Zugriffe übersteigt. Man muss also abschätzen, ob der Einsatz von Replikaten den zusätzlichen Aufwand (Traffic, Resourcen) rechtfertigt (trade-off).



Das Content Placement unterscheidet die 3 folgenden Arten von Replikas: permanent, Server-initiiert sowie Client-initiiert.

Permanente Replikas kann man sich als bereits von Anfang an vorhandenes, statisches kleines Set von Kopien vorstellen, die zusammen einen verteilten Speicher darstellen. Beispiele dafür sind das Mirroring von Webseiten, bei dem diese auf unterschiedliche Server (geographisch) skaliert werden, sowie die Replikation von Daten in Server-Clustern, wodurch jeder Rechner seinen eigenen Datenbestand erhält.
Server-initiierte Replikas dienen der Leistungssteigerung und werden vom Besitzer der Daten (Server) veranlasst. Beispielsweise hat ein Web-Server über einen bestimmten Zeitraum eine große Menge von Anfragen einer „exotischen“ Location zu bedienen. Dann ist es sinnvoll, temporäre Replikas dynamisch in dieser Umgebung zu installieren.
Client-initiierte Replikas werden auch als *Caches* bezeichnet. Dabei wird vom Client *temporär* eine Kopie der von ihm zuvor abgefragten Daten angelegt, wodurch sich in vielen Fällen die Zugriffszeit reduziert. Gespeichert werden die replizierten Daten in einem Speicherplatz (Cache), der lokal zum Client ist. Das kann am selben Rechner oder aber auf einer anderen Maschine im lokalen Netzwerk sein. Somit können sich auch mehrere Rechner einen einzigen Cache teilen. Führt der Client eine Anfrage aus, und kann sie vom Cache erfüllt werden, bezeichnet man dies als *cache hit*.

2. **Geben Sie verschiedene Möglichkeiten der Update Propagation (Content Distribution) an und bewerten Sie diese hinsichtlich Vor- und Nachteilen sowie Einsatzmöglichkeiten.**

Update Propagation bezieht sich darauf, wie replizierte Inhalte im Bedarfsfall upgedatet werden. Zunächst stellt sich die Frage *was* überhaupt weitergegeben werden soll.

- *Benachrichtigung*: In einem Invalidierungsprotokoll werden andere Kopien darüber informiert, dass eine Aktualisierung stattgefunden hat und die Daten somit nicht mehr gültig sind. Da keine Datensätze verschickt werden, wird nur wenig Bandbreite im Netzwerk benötigt. Dieses Modell ist dann sinnvoll, wenn viele Aktualisierungen wenigen Lese-Zugriffen gegenüber stehen.

- *Datentransfer*: Sendet die aktualisierten Daten an die Replika-Hosts. Einsatz findet diese Variante bei einem hohen lese-schreib Verhältnis. Andernfalls kann es sein, dass Updates von niemandem gelesen werden und somit obsolet sind. Alternativ ist es möglich, dass nicht Daten, sondern Logs verschickt werden, um Bandbreite zu sparen.
- *Operationstransfer*: Hier werden keine Daten verschickt, sondern jeder Replika mitgeteilt, welche Update-Operationen auf ihren Daten durchzuführen sind. Statt ganzen Dateien werden somit Parameter-Werte versendet. Dieses Modell der Content Distribution wird auch als *aktive Replikation* bezeichnet, da angenommen wird, dass jede Kopie über einen Prozess verfügt, der sie „aktiv“ auf dem aktuellen Stand halten kann. Vorteil ist hier eine geringe Bandbreiten-Anforderung, allerdings kann es bei komplexen Operationen zu einem hohen lokalen Rechenaufwand kommen.

Push/Pull Protokolle:
Dieser Entwurfsaskpekt unterscheidet, ob Aktualisierungen abgeholt (pull) oder automatisch verschickt werden (push). In *push*-basierten Ansätzen, auch als Server-basierte Protokolle bezeichnet, werden Aktualisierungen an andere Replikas weitergegeben, ohne dass diese welche angefordert haben. Dies wird besonders in Fällen eingesetzt, wo Replikas im Allgemeinen einen relativ hohen Konsistenzgrad aufweisen müssen, etwa beim Caching, wo v.a. Leseoperationen durchgeführt werden. Charakteristisch ist auch, dass am Server ein hoher Overhead entstehen kann, da alle zu aktualisierenden Client-Caches von ihm verwaltet werden müssen (*stateful Server*).

In einem *pull*-basierten Ansatz, auch als Client-basiert bezeichnet, fordert ein Server oder Client einen anderen Server auf, ihm Aktualisierungen zu senden, die zu diesem Zeitpunkt vorliegen. Dieser Ansatz wird häufig von Web-Caches verwendet, wo der Browser zuerst prüft, ob die im Cache befindlichen Datenelemente noch aktuell sind, und dann bei Bedarf die Aktualisierung vom Webserver holt. Cient-basierte Protokolle sind effizient, wenn das Lese/Aktualisierungs Verhältnis relativ niedrig ist. Der größte Nachteil ist, dass die Antwortzeit im Falle eines veralteten Cache-Eintrages steigt.

Issue	Push-based	Pull-based
State at server	List of client replicas and caches	None
Messages sent	Update (and possibly fetch update later)	Poll and update
Response time at client	Immediate (or fetch-update time)	Fetch-update time

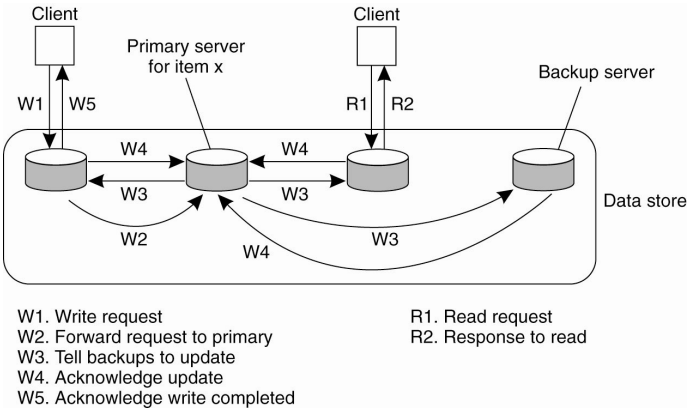
Uni-/Multicasting:
In Zusammenhang mit der Einscheidung nach push/pull Protokoll muss geklärt werden, ob *uni-* oder *multicasting* verwendet werden soll. Bei einer Unicast-Kommunikation sedet ein Server, der Teil des Datenspeichers ist, seine Aktualisierung an N andere Server, indem er N separate Nachrichten sendet, je eine an jeden Server. Beim Multicasting übernimmt das zugrunde liegende Netzwerk die Aufgabe, eine Nachricht effizient an mehrere Empfänger zu senden. Das ist dann von Vorteil, wenn alle Replikas im selben LAN hängen und somit auf das Hardware-Broadcasting zurück gegriffen werden kann. *Multicasting* kann häufig effizient mit einem *push*-basierten Ansatz kombiniert werden.In diesem Fall verwendet ein Server eine Multicast-Gruppe, um mehreren anderen Server (welche in der Multicast-Gruppe sind) Aktualisierungen bereitzustellen. Bei einem *pull*-basiertem Ansatz ist es häufig nur ein einziger Client der eine Aktualisierung anfordert. In diesem Fall wird *Unicasting* die effizientere Lösung sein.

3. **Erläutern Sie die Funktionsweise der "primary-based" Protokolle. Bewerten und vergleichen Sie die verschiedenen Arten.**

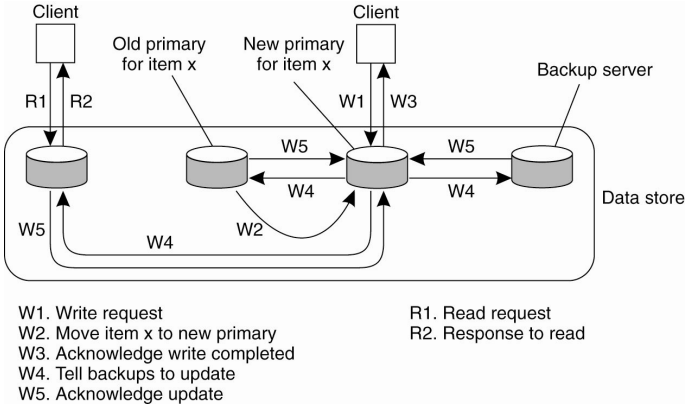
Primary-based Protokolle beschreiben eine Art der Implementierung von Konsistenz-Modellen. Dabei werden alle Update-Oparationen zu einer Primärkopie weitergeleitet, die ihrerseits sicherstellt, dass Updates korrekt weitergegeben und in der richtigen Reihenfolge durchgeführt werden. Im Gegensatz dazu werden bei Replikated-Write Protokollen Updates an mehrere Replikas zugleich gesendet. In primary-based Protokollen wird jedem Datensatz x ein Primary zugeordnet, der für die Koordination der Schreibvorgänge auf x zuständig ist.

Remote-write
Bei dem einfachsten Primary-based protokoll werden alle Schreib-Operationen zu einem bestimmten (fixierten) Primary Server weitergeleitet. Dieser führt daraufhin ein Update auf der lokalen Kopie des Items durch und leitet es an alle Backup-Server weiter, die ihrerseits lokale Updates durchführen und abschließend ein Acknowledge zurück an den Primary senden. Wenn alle Aktualisierungen durchgeführt, wird dies vom Primary dem initialisierenden Prozess bestätigt. Das performance Problem bei diesem Schema ist, dass es relativ lange braucht, bevor ein Prozess, der ein Update initiiert hat, fortfahren darf. Eine Alternative wäre

einen nonblocking Ansatz zu wählen, bei dem aber nicht sichergestellt werden kann, dass die Updates von den Backup Servern durchgeführt wurden.



Local-write
Um ein Update an einem Item durchzuführen, wird zuerst die primäre Kopie lokalisiert (wie zuvor) und das eigene File mit dieser Kopie überschrieben. Der Hauptvorteil dieser Vorgehensweise besteht darin, dass mehrere Schreiboperationen nacheinander lokal erfolgen können, während lesende Prozesse weiterhein auf ihre lokalen Kopien zugreifen können. Eine Verbesserung dieser Art kann jedoch nur erreicht werden, indem non-blocking Protokolle verwendet und Aktualisierungen an die Replikate weitergeleitet werden, nachdem der primäre Server das lokale Update beendet hat.

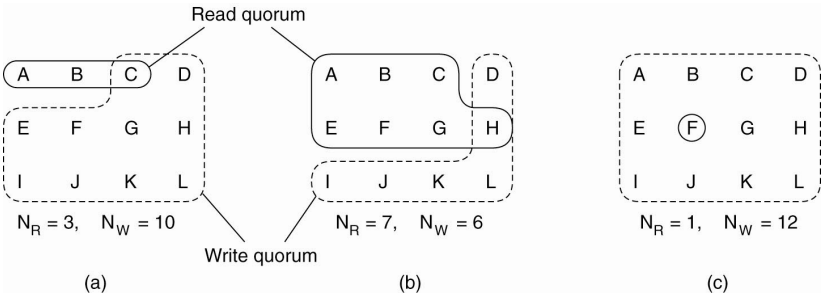


4. **Erläutern Sie die Funktionsweise der "replicated-write" Protokolle. Bewerten und vergleichen Sie die verschiedenen Arten. Welche Probleme können bei "Active Replication" auftreten? Erklären Sie "quorum-based" Replikationsprotokolle und geben Sie die Bedingungen an, um Read-Write und Write-Write Konflikte zu verhindern.**

Replicated-write Protokolle unterstützen parallele Schreib-Operationen an mehreren Kopien. Dabei wird zwischen aktiver Replikation, in der Operationen an alle Replikas weitergeleitet werden, und Konsistenz-Protokollen, die auf einer Mehrheitsentscheidung (Quorum) basieren, unterschieden.

Aktive Replikation
Jeder Kopie ist ein Prozess zugeordnet, der Updatevorgänge durchführt, indem er die benötigten Operationen (bzw. eine aktuelle Kopie) an alle Replikas sendet. Dabei muss aber sichergestellt werden, dass alle Operationen überall in der selben Reihenfolge ausgeführt werden (zB mit Lampert Clocks). Besser ist aber die Integration eines zentralen Koordinators, eines *Sequencers*, der allen Operationen eine unique ID zuweist, bevor sie nacheinander, entsprechend ihrer Numerierung, ausgeführt werden. Ein mögliches Problem ist, dass einzelne Requests von einem replizierten Objekt (d.h. von jedem Replikat) mehrfach durchgeführt werden könnten → siehe Replicated Invocations (Frage 5).

Quorum-basierte Replikation
Die Grundidee ist hier, dass Clients eine Erlaubnis für jeden Lese-/Schreibzugriff auf eine Kopie einholen müssen. Möchte der Client eine Kopie updaten, müssen dem mindestens die Hälfte aller Server zustimmen. Erst dann wird die Datei verändert und mit einer neuen Versionsnummer versehen. Damit können write-write Konflikte vermieden werden. Vor Lesezugriffen muss der Client ebenfalls zumindest N/2 + 1 Server kontaktieren. Verfügen alle über die selbe Versionsnummer, muss es sich um die aktuelle Version handeln.

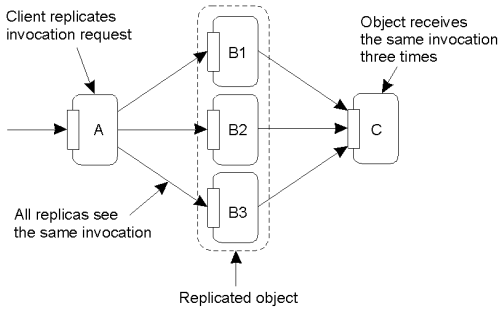


Bedingungen
1) $N_R + N_W > N$
2) $N_W > N/2$

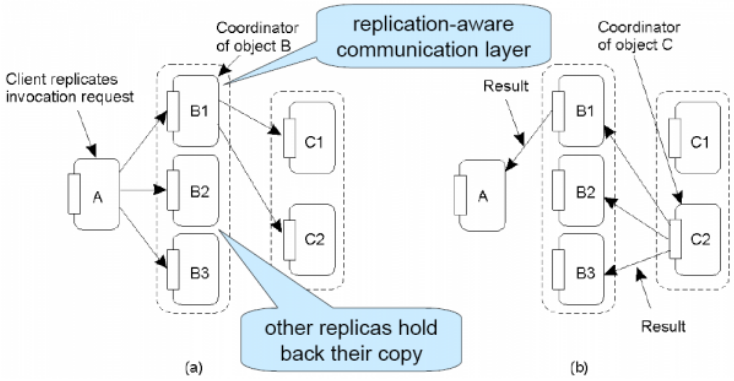
In obenstehender Grafik erzeugt (b) möglicherweise write-write Konflikte, da 2) verletzt ist.

5. **Welche Besonderheiten sind bei der Replikation von Objekten zu beachten? Erläutern Sie (inkl. genauer Skizze) wie man "Replication transparency" in Objektsystemen umsetzen könnte ("replicated invocation").**

Wie in 4) erwähnt, kann es zum Problem von wiederholten Aufrufen kommen, wie in folgender Grafik dargestellt ist.



Eine Möglichkeit ist, Mehrfachaufrufe einfach zu verbieten, was jedoch zu Lasten der Fehlertoleranz geht. Stattdessen wird die folgende Methode gewählt: Nachdem alle Objekte B von A aufgerufen wurden, wird allen Requests von B nach C der selbe UID zugeordnet. Eine der Replikas von B, die als Koordinator fungiert, leitet ihre Anfrage nun an alle Kopien von C weiter, während die anderen Aufrufe von B zurückgehalten werden. Für die Rückleitung der Antwort der Objekte C wird der selbe Mechanismus angewendet. Ein Koordinator von C sendet die Antwort an alle B's, von wo das Resultat vom B-Koordinator nach A geschickt wird.



6. **Was sind Epidemic Protocols. Welche Vor- und Nachteile haben diese? Erklären Sie "gossiping" ("rumor spreading") im Zusammenhang mit Replica update propagation. Erläutern Sie Vor- und Nachteile. Erklären Sie das Anti-Entropy Modell im Zusammenhang mit Replica update propagation. Erläutern Sie Vor- und Nachteile.**

Epidemic Protocols sind für Datenspeicher gedacht, welche nur eine eventuelle Konsistenz aufweisen müssen also durchaus Inkonsistenzen während der Updates beinhalten. Gibt es keine Aktualisierung, muss nur sichergestellt sein, dass alle Repliken irgendwann identisch sind. Weiters sollen Aktualisierungen mit möglichst wenigen Nachrichten an alle Replikas weitergegeben werden. Dadurch wird ein schneller, dezentraler (auf lokalen Informationen basierender) Informationsaustausch innerhalb großer Netzwerke

sichergestellt. Vorteil ist neben der geringen Netzwerkbelastung vor allem die gute Skalierbarkeit. Schwierig ist hingegen das Löschen eines Datensatzes, das an alle übrigen Nodes weitergegeben werden muss. Auch wird, wie bereits erwähnt, keine totale Konsistenz garantiert.

Das *Anti-Entropie Modell* ist ein Weitergabemodell für Epidemische Protokolle welches per Zufall einen anderen Server auswählt und mit diesem dann Aktualisierungen austauscht (push bzw. pull). Sind die meisten Knoten „infiziert“, ist beispielsweise die Chance relativ gering über den push-Ansatz weitere Knoten für ein update zu finden.

Gossiping ist eine spezielle, effizientere, Form des Anti-Entropy Modells. Die Funktionsweise ist einfach: Wenn ein Datenelement aktualisiert wurde, wendet sich der Server an einen beliebigen anderen Server um ihm die "Neuigkeit" mitzuteilen. Dieser wiederum macht dasselbe und kontaktiert den nächsten Server um die Aktualisierung vorzunehmen usw. Wenn ein Server erreicht wird, der schon "infiziert" wurde, gibt es nur eine geringe Wahrscheinlichkeit der Weiterleitung. Zumeist wird die Nachricht in diesem Fall gelöscht. Gossiping ermöglicht ein rasches „Spreading“, allerdings kann nicht garantiert werden, dass alle Server benachrichtigt werden.