Consistency and Replication

40. 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.

Bei Replikation führt man mehrere Kopien einer Entität auf verschiedenen Knoten.

Hauptgründe für Replikation:

- **Zuverlässigkeit:**
  - Umschalten im Fehlerfall --> **Fehlertoleranz**
  - Schutz gegen beschädigte Daten (corrupted data)
- **Leistung durch Skalierbarkeit:**
  - **Größe**: Die Skalierung nach der Größe tritt zB auf, wenn immer mehr Prozesse auf Daten zugreifen müssen, die von einem einzigen Server verwaltet werden. In diesem Fall kann die Leistung verbessert werden, indem man diesen Server repliziert und damit die Arbeit aufteilt.
  - **geographischer/topologischer Kontext**: Die Skalierung in Hinblick auf die Größe eines geografischen Bereichs kann ebenfalls eine Replikation erforderlich machen. Dabei wird eine Kopie der Daten in die Nähe des Prozesses platziert, der sie nutzt, und somit die Datenzugriffszeit reduziert.

Skalierbarkeit und Replikation:

Replikation bedeutet deutlichen Mehraufwand, da die Daten konsistent gehalten werden müssen. Man muss also abschätzen, ob das Einsetzen von Replikaten den zusätzlichen Aufwand (Traffic, Ressourcen) rechtfertigt. (Aufwand größer als Nutzen?)

Content Replication und Content Placement:

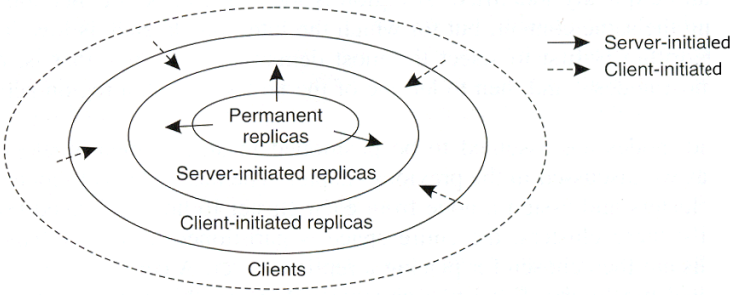


Figure 7-17. The logical organization of different kinds of copies of a database into three concentric rings.

- **Content Replication:** Sind Replikate permanent vorhanden oder nur zur Laufzeit
- **Content Placement:** Wo werden die Replikate eingerichtet? (Client oder Server)

- **Permanente Repliken:**
  - Cluster Based: Server befinden sich am selben Ort, es kann z.B. Round Robin durchgeführt werden
  - Mirroring: gespiegelte Server die sich an einem anderen Ort befinden
- **Server-initiierte Repliken:**
  - Replikat wird dynamisch initialisiert, wenn es zu vielen Requests kommt. Dazu ist es notwendig, die Anzahl der Zugriffe auf ein File zu zählen und zu wissen, woher der Zugriff kommt
  - Schwellenwert für Replikat erstellen (ab wann wird ein Replikat erstellt)
  - Replikat wird gelöscht, wenn es nicht mehr benötigt wird (wobei es immer min. ein Replikat geben muss)
  - Replikat wird möglichst dort erstellt, wo die meisten Requests herkommen
  - Ideal, um flash crowds abzufangen. WP: Flash crowd describes a network phenomenon where a network or host suddenly receives a lot of traffic. This is due to the appearance of a web site on a blog or news column.
- **Client-initiierte Repliken:** Auch bekannt als Cache, werden auf dem lokalen Speicher oder im lokalen Netzwerk platziert. Dienen zum Verbessern der Zugriffszeiten auf Daten, werden jedoch nur eine begrenzte Zeit gespeichert, um zu verhindern, dass extrem veraltete Daten verwendet werden. Effizienz kann gesteigert werden, wenn ein gemeinsamer Cache verwendet wird.

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

Bei der Update Propagation von Replikas ist es nötig, verschiedene Aspekte zu betrachten und gegeneinander abzuwiegen.

Status versus Operation:

Ein wichtiger Entwurfsaspekt kümmert sich darum, was überhaupt weitergegeben werden soll. Dafür gibt es grundsätzlich 3 Möglichkeiten:

- **Weitergabe der Aktualisierung nur durch eine Benachrichtigung:** In einem Invalidierungsprotokoll werden andere Kopien darüber informiert, dass eine Aktualisierung stattgefunden hat und dass ihre Daten nicht mehr gültig sind. Erst bei einem Lesezugriff werden diese Daten aktualisiert.
  - Vorteil: wenig Netzwerkbandbreite verbraucht, da nur eine Benachrichtigung versendet wird.
  - Einsatzbereich: wenn es viele Aktualisierungsoperationen im Vergleich zu Leseoperationen gibt.
- **Übertragung der Aktualisierungs-Daten**
  - Nachteil: Viel Bandbreitenverbrauch
  - Einsatzbereich: dort wo das Lese/Schreib Verhältnis hoch ist.
- **Weitergabe der Aktualisierungs-Operation**: Hier werden keine Änderungen an den Daten übertragen, sondern jeder Replik wird mitgeteilt, welche Aktualisierungsoperation sie ausführen soll. Dieser Ansatz wird auch aktive Replikation genannt. (zB SQL Statements bei Oracle Standby-Server).
  - Vorteil: Aktualisierungen können mit minimalen Kosten für die Bandbreite weitergegeben werden.
  - Nachteil: Jede Replik benötigt möglicherweise mehr Rechenleistung, insbesondere wenn die Operationen relativ komplex sind.

Pull- oder Push-Protokolle:

Dieser Entwurfsaspekt unterscheidet sich darin, ob Aktualisierungen abgeholt (pull) oder automatisch verschickt werden (push).

- **In einem push-basierten Ansatz,** auch als Server-basierte Protokolle bezeichnet, werden Aktualisierungen an andere Repliken weitergegeben, ohne dass diese welche angefordert haben. Server-basierte Protokolle werden hauptsächlich dort eingesetzt, wo die Repliken im Allgemeinen einen relativ hohen Konsistenzgrad aufweisen müssen. Dieser Bedarf eines hohen Konsistenzgrades hat mit der Tatsache zu tun, dass permanente und Server-initiierte Repliken sowie große Caches häufig von vielen Clients gemeinsam genutzt werden, die wiederum hauptsächlich Leseoperationen durchführen. Daher ist das Lese/Aktualisierung Verhältnis relativ hoch (eine Aktualisierung wird oft gelesen**). Ein wichtiger Aspekt** bei push-basierten Ansätzen ist, dass der Server alle Clients verwalten muss, welche Aktualisierungen von ihm erhalten. Neben der Tatsache, dass statusbehaftete Server weniger fehlertolerant sind, kann die Verwaltung aller Client-Repliken und -Caches einen wesentlichen Overhead auf dem Server verursachen
- **In einem pull-basierten Ansatz,** auch als Client-basierte Protokolle bezeichnet, fordert ein Server oder Client einen anderen Server auf, ihm Aktualisierungen zu senden, die ihm 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. Client-basierte Protokolle sind effizient, wenn das Lese/Aktualisierung Verhältnis relativ niedrig ist. Der größte Nachteil ist, dass die Antwortzeit im Falle eines veralteten Cache-Eintrag steigt.

Aspekt	**Push-basiert**	**Pull-basiert**
Status am Server	Liste mit Client-Repliken und Caches	Keine
Gesendete Nachrichten	Aktualisierung (und möglicherweise später die Aktualiserung laden)	Pull und Aktualisieren
Antwortzeiten auf dem Client	Unmittelbar	Laden/Aktualiserungszeit

Häufig wird eine Mischform aus beiden Ansätzen verwendet, die auf **Leases** basiert. Ein Lease ist eine Zusage des Servers, dass er dem Client eine bestimmte Zeit lang Aktualisierungen bereitstellt. Wenn ein Lease abläuft, ist der Client gezwungen, den Server für mögliche Aktualisierungen zu kontaktieren, um diese bei Bedarf per Pull zu erhalten.

Wie die Expiration time bei Leases gewählt wird:

- **Age-based:** Annahme, dass länger unveränderte Daten auch weiterhin wenig Änderungen haben werden. Daten mit länger zurückliegender Änderungszeit bekommen höhere Expiration time.
- **Renewal-frequency-based:** Clients, die eine hohe Update Rate haben bekommen, eine höhere Expiration time

- **State-space overhead:** Schutzmechanismus um eine Überlastung des Servers zu vermeiden. Wenn der Server eine Überlastung feststellt verringert er die Expiration time der Leases und reguliert somit seine Last.

Unicasting versus Multicasting:

- Bei einer Unicast-Kommunikation sendet 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 Multicasting kann häufig effizient mit einem push-basierten Ansatz kombiniert werden, um Aktualisierungen weiterzugeben. In diesem Fall verwendet ein Server eine Multicast-Gruppe, um mehrere andere 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.

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

Primary Based Protokolle:

In Primary Based Protokollen ist jedem Datenelement x im Speicher ein primärer Server zugeteilt, der für die Koordination von Schreiboperationen für x verantwortlich ist. Ein Schreibrequest auf einem beliebigen anderen Replikat wird an den Primary-Server weitergereicht (remote-write). Geschrieben wird immer nur auf einem Replikat. Der Primary kann dabei für jedes Datenitem ein anderer Host sein. Alternativ dazu gibt es noch die Möglichkeit, das Datenitem zuerst auf den Server zu migirieren, auf dem derSchreibzugriff abgesetzt wird (Änderung des Primarys) und die Operation dann dort auszuführen (localwrite).

Remote-write:

Bei dem einfachsten Primary-based Protokoll werden alle Operationen zu einem bestimmten (fixierten) Single-Server weitergeleitet. Wenn z.B. eine Schreiboperation stattfinden soll, wird die Operation zuerst weitergeleitet an den Primary Server für das Item. Der Primary Server führt daraufhin ein Update auf der lokalen Kopie des Items durch und leitet das Update an alle Backup-Server weiter. Jeder Backup-Server führt das Update genauso durch und sendet ein Acknowledge zurück an den Primary Server. Wenn alle aktualisiert wurden, schickt der Primary Server ein Acknowledge zurück zu dem initialisierenden Prozess. 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. Das Problem bei einem nonblocking System ist die Fehlertoleranz. Der Vorteil ist die Geschwindigkeit.

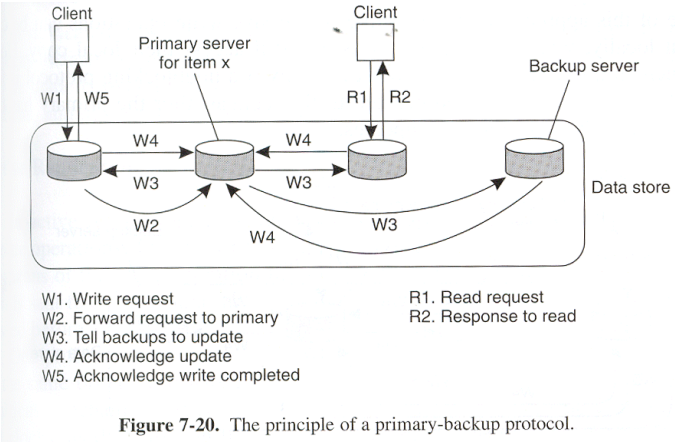


Figure 7-20. The principle of a primary-backup protocol.

Local-write:

Wenn ein Item aktualisiert werden soll, wird zuerst die primäre Kopie lokalisiert (wie zuvor). Und dann wird diese Kopie auf die eigene location kopiert. dazu gibt es noch die Möglichkeit, das Datenitem zuerst auf den Server zu migrieren, auf dem der Schreibzugriff abgesetzt wird (Änderung des Primarys) und die Operation dann dort auszuführen (local-write). Der Hauptvorteil dieser Vorgehensweise besteht darin, dass mehrere Schreiboperationen nacheinander lokal erfolgen können, während lesende Prozesse weiterhin auf ihre lokalen Kopien zugreifen können. eine Verbesserung dieser Art kann jedoch nur dann erfolgen, wenn ein nicht sperrendes Protokoll beachtet wird, das Aktualisierungen an die Replikate weiterleiten, nachdem der primäre Server sie lokal erfolgreich ausgeführt hat.

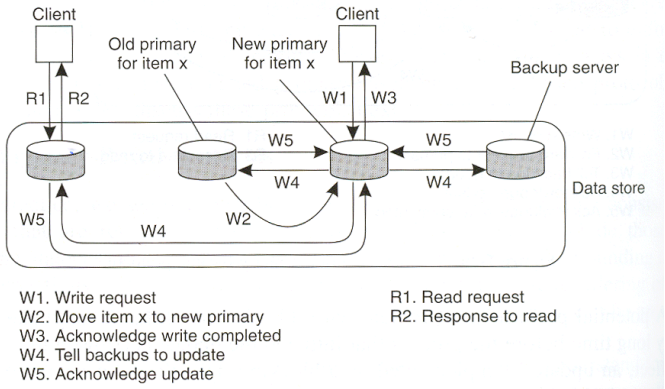


Figure 7-21. Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

43. 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:

Im Gegensatz zu Primary-Based Protokollen erlauben Replicated-Write-Protokolle den Zugriff auf beliebige Replikate. Die Änderungen werden dann an alle anderen Replikate propagiert, entweder über:

- **Active Replication:** Die Operation wird an jedes Replica gesendet. Das Problem ist, dass Operationen überall in der selben Reihenfolge geliefert werden müssen. Es wird daher ein "totally-ordered multicast- Mechanismus" benötigt. Hier gibt es aber in groß skalierten Systemen Probleme. Als Alternative kann ein zentraler Koordinator (Sequencer) verwendet werden. Meistens ist eine Mischung aus beiden nötig.
- **Quorum-Based Protocols**: Hier wird ein Voting verwendet. Bsp.: Um ein File zu aktualisieren, muss der Client zuerst mindestens die Hälfte der Server plus 1 dazu bringen, dem Update zuzustimmen. Um das File zu lesen, muss der Client wiederum mindestens die Hälfte der Server plus eins dazu bringen, ihre Versionsnummern des Files zu schicken. Wenn alle Versionsnummer gleich sind, muss es sich um die aktuelle Version handeln.
- **Gifford's scheme** : Eine Verallgemeinerung hierfür ist das **Gifford's scheme**, das folgenden Regeln folgt: Der Client muss eine Mehrheit der Server mit dem Update versorgen. Beim Lesezugriff muss auch wieder von zumindest der Anzahl der nicht upgedateten Server +1 gelesen werden. Dabei gewinnt die Dateneinheit mit der höchsten Versionsnummer.

  Nr + Nw > N (verhindert read-write Konflikte)

  Nw > N/2 (verhindert write-write Konflikte)

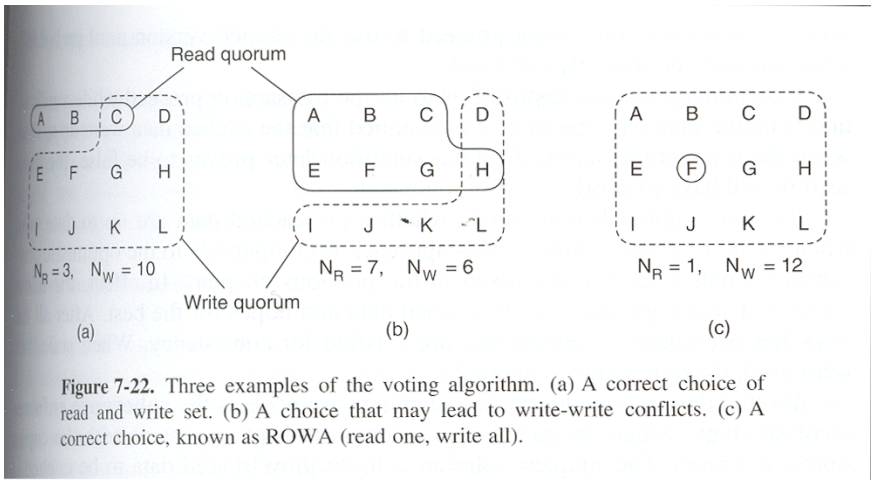


Figure 7-22. Three examples of the voting algorithm. (a) A correct choice of read and write set. (b) A choice that may lead to write-write conflicts. (c) A correct choice, known as ROWA (read one, write all).

- **Coordinator-Cohort-Replication**: Eine Spezialform von Replicated-Write-Protokollen ist **Coordinator-Cohort-Replication**. Dabei wird die Anfrage an ein (beliebiges) Replikat gesendet, das die Updates synchron an alle anderen propagiert. Dazu ist ein distributed-Locking-Mechanismus erforderlich.

44. 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").

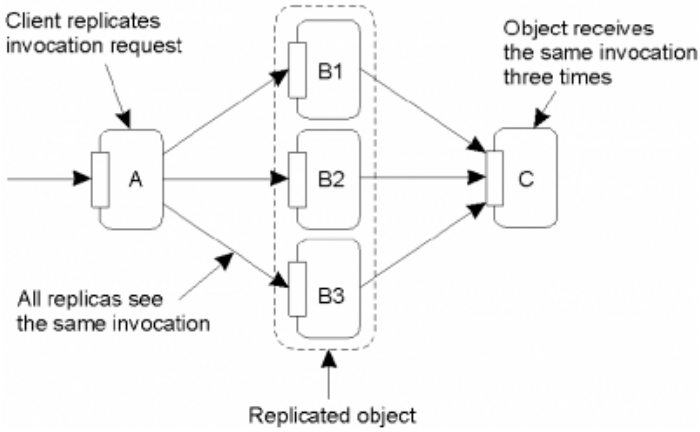
Problem: Wenn Objekte in verteilten Systeme verwendet werden, und ebenfalls verteilt sind, dann müssen auch die Zugriffe auf Objekte konsistent gehalten werden, da es sonst zu inkonsitenten Zuständen innerhalb des Systems führen kann (entry consistency). Wir müssen also eine gleichzeitige Ausführung von Methoden am Objekt verhindern (durch Locking relativ einfach zu lösen) und wir müssen Änderungen (Methodenaufrufe) auf ALLE Replikate des Objektes verteilen um sicherzustellen, dass nicht 2 unterschiedliche Aufrufe auf ein verteiltes System zur gleichen Zeit geschehen.

Dies kann folgendermaßen geschehen:

- **primary-based approach**: ein "Koordinator" übernimmt die Kontrolle und delegiert die Aufrufe. Problem: Overhead, Skalierbarkeit geht verloren, "Koordinator" muss andere verteilte Objekte kennen.
- **totally-ordered-multicasts**: Methodenaufrufe werden zB durch Lamport clocks durchnummeriert und dann in der Reihenfolge auf jedes Objekt ausgeführt. Problem dabei: sehr aufwändig, viel Overhead

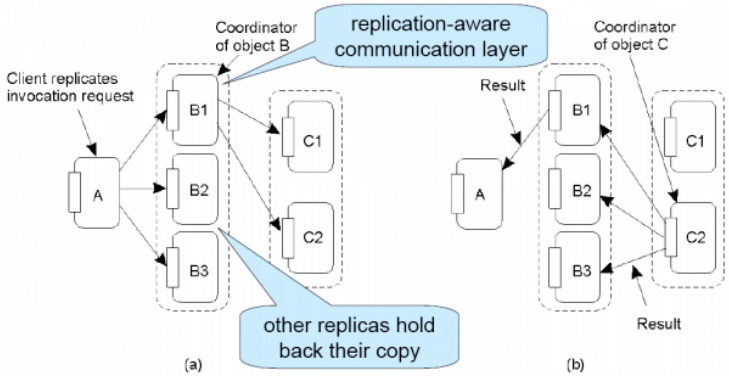
Replicated Invocation:

Problem: Methodenaufrufe auf ein vielleicht nicht verteiltes Objekt werden mehrmals ausgeführt.



Lösung: Realisierung von **Replication Transparency**:

Falls von A aus ein Aufruf in B erfolgt, so ruft A alle Replikate in B (B1, B2, B3) auf. Wenn B nun seinerseits wieder eine Methode von C aufruft, so wird das nur vom Koordinator von B (z.B. B1) gemacht (und zwar bei C1 und C2). Andernfalls (wenn alle 3 Replikate von B den Aufruf durchführen würden) wäre die Methode von C zu oft aufgerufen worden. Die Antworten werden nur vom Koordinator zurückgegeben, also von z.B. C2 zu allen 3 Replikaten von B, und von B1 zu A.



45. 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:

Epidemic Protocols sind für Datenspeicher gedacht, welche nur eine „schlußendliche" Konsistenz aufweisen müssen. Das heißt: Wenn es eine Aktualisierung gibt, muss nur sichergestellt sein, dass alle Replikas irgendwann identisch sind.

Das Hauptziel dieser Protokolle ist es, schnell Information an viele Knoten zu verbreiten, während man nur lokale Informationen verwendet.

- Vorteile: Gute Skalierbarkeit, Aktualisierung an alle Repliken erfolgt in so wenigen Nachrichten wie möglich -> geringe Netzwerkbelastung.
- Nachteile: Weitergabe des Löschens eines Datenelements ist schwierig (Death certificates), löst keine Aktualisierungskonflikte, nur schlußendliche Konsistenz (sehr schwache Konsistenz).

Anti-Entropy Modell:

- Das **Anti-Entropy Modell** ist ein Weitergabemodell für Epidemische Protokolle, welches per Zufall einen anderen Server auswählt, und mit diesem dann Aktualisierungen austauscht. Es kann pull- oder pushed basiert arbeiten (Ein Server schickt seine Updates zu einem anderen, oder holt sie von einem anderen). Wenn viele Knoten infiziert sind, ist die Chance relativ gering über den push-Ansatz weitere Knoten für ein Update zu finden, im Gegensatz zum Pull-Ansatz.

Gossiping:

- **Gossiping** oder auch „**rumor spreading**" genannt ist eine spezielle Form des Anti-Entropy Modells und ein effizientes Weitergabemodell für Epidemische Protokolle. 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, ist die "Neuigkeit" nicht mehr so interessant und macht nur mehr mit einer gewissen Wahrscheinlichkeit weiter mit "gossiping".
  - Vorteil: Aktualisierung wird schnell weitergereicht
  - Nachteil: Es kann nicht garantiert werden, dass alle Server "infiziert" werden, sprich mit Aktualisierungen versorgt werden.