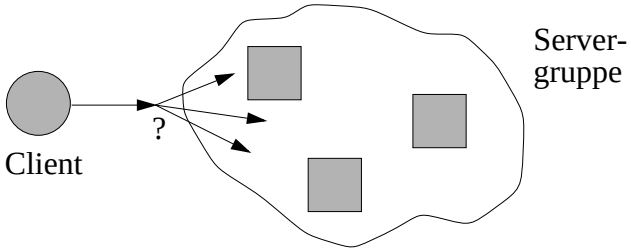
Serverwahl bei einem Lastverbund

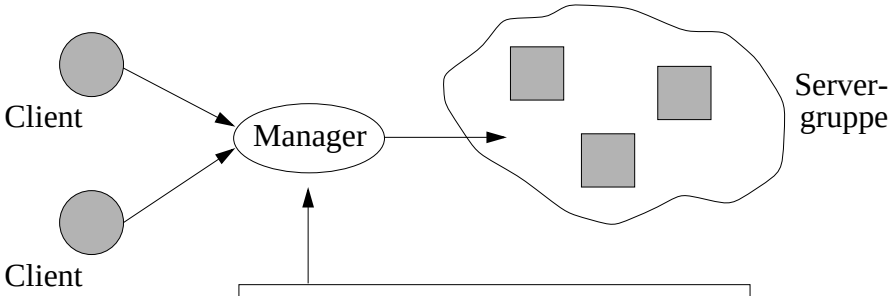


1) *Zufallsauswahl*

- Einfaches, effizientes Protokoll
- Nachteile:
  - Client muss mehrere Server kennen
  - ggf. ungleichmässige Auslastung

Stellen Verfahren mit "round robin"-Einträgen im DNS-System eine solche Zufallsauswahl dar?

2) *Zentraler Service-Manager*

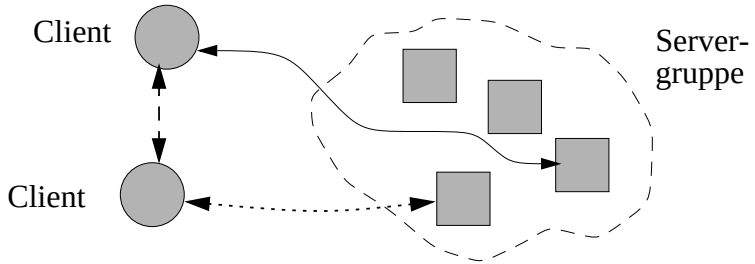


- sorgt für sinnvolle Verteilung (wie?)
- behält ggf. Überblick über Aufträge
- informiert sich ggf. von Zeit zu Zeit über die Server-Lastsituation

- Nachteile:
  - Overhead bei trivialen Diensten
  - ggf. Überlastung des Managers
  - Dienstblockade bei Ausfall des Managers

Serverwahl bei Lastverbund (2)

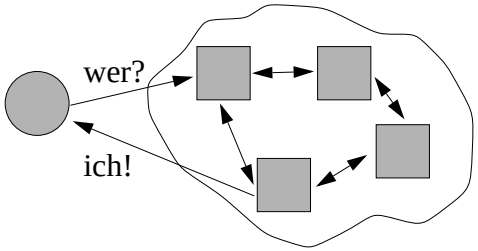
3) *Clients einigen sich untereinander*



- u.U. grosser Kommunikationsaufwand zwischen vielen Clients
- Clients kennen sich i.a. nicht (z.B. bei dynamisch gegründeten)

4) *Server einigen sich untereinander*, wer den Auftrag ausführt

- Election-Protokoll (aber fehlertolerant wegen möglichen Server-Ausfällen)
- ggf. Kooperations-topologie festlegen



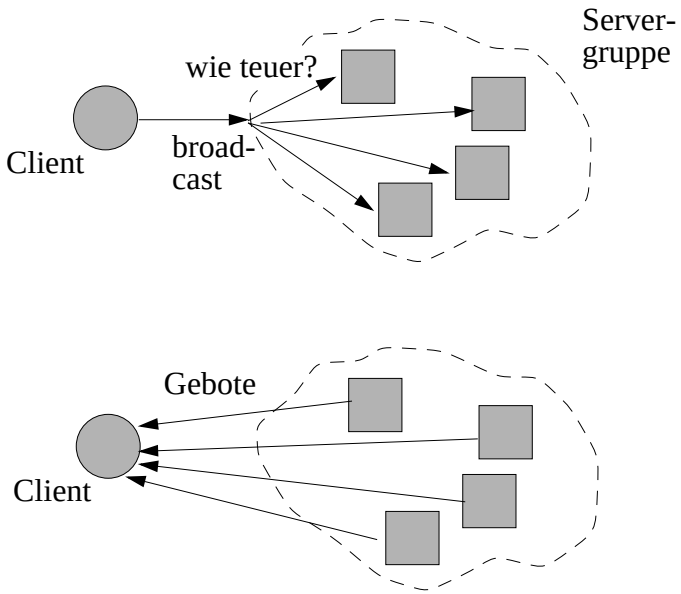
- i.a. nur wenige Server (relativ zur Zahl der Clients)
- Server führen Abstimmung diszipliniert durch (verlässlicher als Clients)

5) "*Anycast*": der nächstliegende (=?) Server wird von der Netzinfrastruktur (Router etc.) bestimmt

Serverwahl bei Lastverbund (3)

6) *Bidding-Protokoll*

- Client fragt per Broadcast nach Geboten
- Server mit “billigstem” Angebot wird ausgewählt



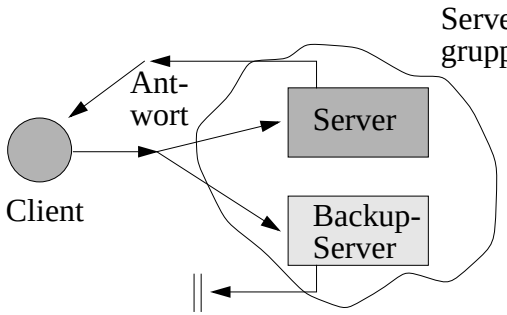
- Variante: nur *Stichprobe* befragen (multicast statt broadcast; sehr kleine Teilmenge von vielen Servern genügt i.a.!)

- Generelles Problem: Lastsituation kann veraltet sein!

Serverreplikation in Überlebensverbunden

1) *Zustandsinvariante Dienste*: im Prinzip einfach - nach Crash anderen Server nehmen...

2) *Zustandsändernde Dienste* (hier “hot stand by”):



aktiver Server

passiver Server im “hot stand-by”-Modus (ggf. mehr als einer)

- im Fehlerfall kann *unmittelbar* auf den Backup-Server umgeschaltet werden
- beachte: Replikation sollte transparent für die Clients sein!
- Auftrag wird per Multicast an alle Server verteilt
- nur die Antwort des aktiven Servers wird zurückgeliefert

Probleme:

- evtl. Subaufträge werden mehrfach erteilt (vom Server und Backup-Server) --> Probleme mit zustandsändernden bzw. gegenseitig ausgeschlossenen Subdiensten
- Reihenfolge der Aufträge muss bei allen Servern identisch sein (--> Semantik von multicast insbesondere bei mehreren Clients)
- Resynchronisation nach einem Crash: Nach Neustart muss ein (passiver) Server mit dem aktuellen Zustand des aktiven Servers initialisiert werden (Zustand kopieren bzw. replay)

Replikation

- Daten mehrfach halten; möglicher Zweck:
  - Effizienzsteigerung: Daten schneller verfügbar machen (z.B. Caches)
  - erhöhte Verfügbarkeit (auch bei Ausfall einzelner Server)
  - Fehlertoleranz durch Majoritätsvotum
- Forderungen: Transparenz und Erfüllung gewisser Konsistenzeigenschaften (“Kohärenz” der Replikate)
- Replikationsmanagement
  - *asynchron*: nur periodische Aktualisierung zwischen den Replikaten (inkohärente Replikate nach Änderung bis zur nächsten Synchronisation)
  - *synchron*: immer kohärente Replikate; logische Sicht eines einzelnen Servers (z.B. hot stand by)
  - Aufwand wächst, je näher man sich dem synchronen Modell annähert
- *Nicht-transparente* Replikation: Client führt Änderungen explizit auf allen Replikaten durch
- *Transparente* Replikation; unterschiedlich realisiert:
  - per Gruppenkommunikation (Semantik und Zuverlässigkeits-aspekte des Kommunikationssystems entscheidend)
  - Hauptserver (“primary”), der Sekundärserver aktualisiert
    - Schreibzugriffe nur beim Primärserver; Lesezugriffe beliebig
    - Hauptserver kann “sofort” oder schubweise (“gossip-Nachricht”) die Sekundärserver aktualisieren (Konsistenzproblematik beachten!)
  - symmetrische Server, die sich jeweils untereinander abgleichen
- Voting-Verfahren: zum Schreiben und Lesen auf jeweils mehr als 1+N/2 Server zugreifen
  - Abgleich (voting) bzgl. neuester Versionsnummer