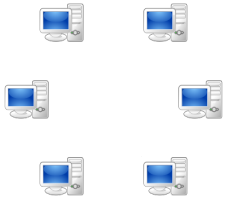
Überblick

Zuverlässige Gruppenkommunikation
- Grundlagen
- Zustellungsgarantien bei Multicast
- JGroups
- Übungsaufgabe 5

Motivation

> *„Das Ganze ist mehr als die Summe seiner Teile."*
>
> Aristoteles

- Problemstellungen gilt es oft im Verbund zu bearbeiten
- Zusammenschluss einzelner Knoten (Rechner, Prozesse) zu Gruppen
- Einheitliches Kommunikationsmittel innerhalb der Gruppe



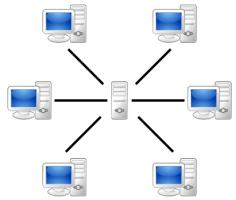
Zusammenschlüsse in verteilten Systemen

- Gruppe: **Zusammenschluss von Knoten** in einem verteilten System
  - Limitierte Anzahl der Knoten
  - Gemeinsamer (globaler) Zustand notwendig
- Szenarien, Anwendungs- und Einsatzgebiete
  - Replikation
  - Instant Messaging
  - Virtuelle Konferenzen
  - Netzwerkanwendungen und -spiele
- Abgrenzung
  - Client-Server-System
  - Peer-to-Peer-System (P2P)

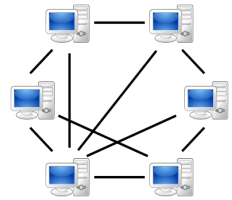
→ In vielen Systemen sind die Grenzen jedoch fließend

Client-Server- und Peer-to-Peer-Systeme

- Client-Server-System
  - Arbeitsteilung
    → Client: Lokale Bereitstellung eines Dienstes
    → Server: Diensterbringer
  - Clients sind unabhängig voneinander
  - Kommunikation ist nicht gleichberechtigt



- Peer-to-Peer-System (P2P)
  - Knoten sind Client *und* Server zugleich
  - Jeder Konten besitzt nur eine partielle Sicht auf das Gesamtsystem
  - Auf sehr große Anzahlen von Knoten ausgelegt



Virtuelle Synchronität (Virtual Synchrony)

- Probleme im Gruppenverbund
  - Kein gemeinsamer Speicher
  - Keine gemeinsame Uhr
  - Zusammensetzung der Gruppe ist oftmals dynamisch
    - Knoten treten ihr bei oder verlassen sie
    - Ausfall von Knoten
    - Abbruch von Verbindungen
- Lösungsansatz: Virtuelle Synchronität (Virtual Synchrony)
  - Knoten einigen sich auf Liste aller aktiven Gruppenmitglieder
    → Gemeinsame Sicht auf das Gesamtsystem (View)
  - Erneutes Aushandeln der View bei Änderung der Zusammensetzung
    → Abfolge von Views dient als gemeinsame logische Zeitbasis
  - Nachrichten sind nur für eine View gültig

Kenneth P. Birman and Thomas A. Joseph
Exploiting Virtual Synchrony in Distributed Systems
Proceedings of the Eleventh ACM Symposium on Operating Systems Principles (SOSP), pages 123–138, 1987.

Überblick

Zuverlässigkeit

- Best-Effort Multicast
  - Versendet ein korrekter Knoten eine Nachricht, wird sie letztendlich von jedem korrekten Knoten ausgeliefert (**Gültigkeit**)
  - Keine Nachricht wird mehrmals ausgeliefert (**keine Verdopplung**)
  - Nur zuvor erstellte und versendete Nachrichten werden ausgeliefert (**keine Erschaffung**)
- Zuverlässiger Multicast
  - Liefert ein korrekter Knoten eine Nachricht aus, wird sie letztendlich von jedem korrekten Knoten ausgeliefert (**Einigkeit**)
  - Ansonsten wie Best-Effort Multicast
- Uniformer Multicast
  - Liefert ein beliebiger Knoten eine Nachricht aus, wird sie letztendlich von jedem korrekten Knoten ausgeliefert (**Uniforme Einigkeit**)
  - Ansonsten wie zuverlässiger Multicast

Ordnung

- Keine Ordnung
  - Nachrichten werden in keiner festen Reihenfolge ausgeliefert
- First-In-First-Out-Ordnung (FIFO-Ordnung)
  - Nachrichten werden von allen korrekten Knoten in der Reihenfolge ausgeliefert, in der sie versendet wurden
  - Von verschiedenen Knoten gesendete Nachrichten werden in keiner festen Reihenfolge ausgeliefert
- Totale Ordnung
  - Nachrichten werden von allen korrekten Knoten in der gleichen Reihenfolge ausgeliefert
  - Totale Ordnung ist orthogonal zur FIFO-Ordnung

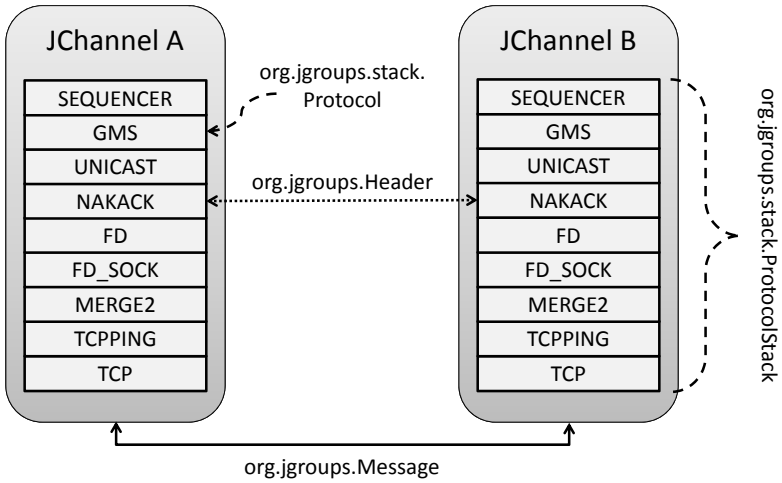
Überblick

JGroups (Kurze Zusammenfassung)

- JGroups
  - Bibliothek und **Framework für zuverlässige Gruppenkommunikation**
    → Virtual Synchrony, Zustandstransfer, Zustellungsgarantie
  - Durch **modularen Aufbau** über Konfiguration an bestehende Erfordernisse anpassbar
- Verwendung
  - Knoten verbinden sich mittels eines `org.jgroups.JChannel`-Objekts
  - Nachrichten (`org.jgroups.Message`) können per Unicast oder Multicast versendet werden
  - Auslieferung von Nachrichten erfolgt asynchron (`org.jgroups.MessageListener`)
  - Benachrichtigung über Gruppenzusammensetzung (`org.jgroups.View`) sind ebenfalls asynchron (`org.jgroups.MembershipListener`)
- Siehe Folien zu Übungsaufgabe 4 (Replikation)
- API-Dokumentation: `http://jgroups.org/javadoc/index.html`

Protokoll-Stack

- Protokoll-Stack von JGroups ist **konfigurier- und erweiterbar**



Protokolle

- Bestehende Protokolle (Auswahl)
  - SEQUENCER
    - Realisiert eine totale Ordnung auf Basis von NAKACK
  - GMS (Group Membership Service)
    - Protokoll für Gruppenmitgliedschaft und Sichten (Views)
  - NAKACK (Not Acknowledge, Acknowledge)
    - Implementiert FIFO-Multicast
  - FD (Failure Detection)
    - Heartbeat-Protokoll für Ausfallerkennung
  - TCP/UDP
    - Transportprotokolle
- Vollständige Liste: `jg-protocol-ids.xml` (→ JGroups Quellcode)
- Implementierung eigener Protokolle möglich
- Protokolle leiten von der Klasse `org.jgroups.stack.Protocol` ab
  - Registrierung mittels XML-Datei oder zur Laufzeit

```
ClassConfigurator.addProtocol(short id, Class protocol);
```

Ereignisse

- Empfang und Versand von Nachrichten sowie Statusänderungen werden als *Ereignisse* im Protokoll-Stack propagiert
  - Zugehörige Methoden der Klasse `Protocol`

```
Object down(Event evt);   // Aufruf durch hoehere Schicht
Object up(Event evt);     // Aufruf durch untere Schicht
```

    - Rückgabe von Ergebnis der unteren (`super.down(evt);`) oder höheren Schichten (`super.up(evt);`) bzw. `null`, wenn Ereignis verworfen wurde
  - Klasse `org.jgroups.Event`

```
int getType();            // Typ des Ereignisses
Object getArg();          // Mitgeliefertes Argument
```

Typ (`Event.*`)	Beschreibung	Argument
`MSG`	Versand (`down`) oder Empfang (`up`) einer Nachricht	`org.jgroups.Message`
`VIEW_CHANGE`	Änderung der aktuellen Sicht (`up` und `down`)	`org.jgroups.View`
`SET_LOCAL_ADDRESS`	Setzen der lokalen Adresse (`down`)	`org.jgroups.Address`

Protokoll-Header

- Protokolle tauschen über *Header* interne Daten zwischen Knoten aus
  - Header leiten von `org.jgroups.Header` ab
  - Müssen vergleichbar zu Protokollen registriert werden

```
ClassConfigurator.add(short id, Class header);
```

  - Header sind Teil von Nachrichten und werden mit diesen übertragen
  - Zugehörige Methoden der Klasse `org.jgroups.Message`
    - Hinzufügen eines Header an Nachricht

```
void putHeader(short id, Header hdr);
```

    - Rückgabe des Header einer Nachricht oder `null`, wenn nicht vorhanden

```
Header getHeader(short id);
```

  - Beispiel:

```
Message msg = new Message(...);
VSTotalOrderHeader hdr = VSTotalOrderHeader.createMulticast(msgid);
msg.putHeader(id, hdr); // id: Attribut aus Oberklasse
```

Serialisierung

- JGroups verwendet eigene Mechanismen zur Serialisierung und Deserialisierung zum Beispiel von Headers
  - Schnittstelle `org.jgroups.util.Streamable`

```
void writeTo(DataOutputStream out);    // Serialisierung
void readFrom(DataInputStream in);     // Deserialisierung
```

  - Klassen müssen über XML-Datei oder zur Laufzeit registriert werden (siehe Registrierung von Headers)
  - Hilfsmethoden in Klasse `org.jgroups.util.Util`

```
byte[] objectToByteBuffer(Object obj);
Object objectFromByteBuffer(byte[] buf, int off, int len);
```

Überblick

Aufgabenstellung

- Implementierung eines **eigenen Sequencers** als JGroups-Protokoll *„VSTotalOrder"*
- Naive Implementierung
  - VSTotalOrder setzt auf NAKACK auf, das FIFO implementiert
  - Umleitung aller Multicast-Nachrichten zu einem ausgewählten Knoten, dem *Leader*
  - Leader versendet Nachrichten

  → Ein einziger Knoten sendet Multicasts + FIFO = totale Ordnung
- Optimierte Implementierung
  - Knoten versenden Multicasts selbst
  - Leader sendet extra Nachricht mit Ordnung
  - Nachrichten werden erst ausgeliefert, wenn Ordnung bekannt

Hinweise zu Teilaufgabe 5.1

- Behandeln von Statusänderungen innerhalb der Gruppe
  - Speichern der lokalen Adresse des Knotens bei `Event.SET_LOCAL_ADDRESS`
  - Speichern der aktuellen Sicht und bestimmen des Leader bei `Event.VIEW_CHANGE`
    - `View` besteht aus geordneter Liste der Adressen aller Mitglieder

      ```
      Vector<Address> view.getMembers()
      ```

    - Erstes Mitglied der aktuellen Sicht ist Leader
    - Zusammen mit lokaler Adresse kann bestimmt werden, ob ein Knoten der Leader ist
- Klasse `VSTotalOrder` (im Pub-Verzeichnis) soll als Grundlage dienen

Hinweise zu Teilaufgabe 5.2 (1/2)

- Umleiten aller Multicasts zu Leader (Naive Implementierung)
  - Überprüfen, ob zu versendende Nachricht außerhalb der Ordnung liegt:

    ```
    msg.isFlagSet(Message.NO_TOTAL_ORDER) ||
    msg.getDest() != null && !msg.getDest().isMulticastAddress()
    ```

    In diesem Fall Ereignis unbehandelt an untere Schicht weiterreichen:

    ```
    super.down(...);
    ```

  - Initialisierung des Versenders der Nachricht mit lokaler Adresse, falls nicht anders festgelegt:

    ```
    msg.getSrc();
    msg.setSrc(...);
    ```

  - Einpacken der Originalnachricht (Serialisierung)
  - Neues Nachrichtenobjekt mit Leader als Empfänger erzeugen, anhängen der Originalnachricht, Header hinzufügen
- Versenden der Nachricht vom Leader
  - Wiederum neues Nachrichtenobjekt erzeugen

    ```
    new Message(null, <local>, msg.getRawBuffer(),
                msg.getOffset(), msg.getLength());
    ```

Hinweise zu Teilaufgabe 5.2 (2/2)

- Nachricht weiterreichen
  - Nachricht vom Leader entgegennehmen
  - Originalnachricht auspacken
  - Originalnachricht an höhere Schicht weiterreichen
- Vorgegebene Klassen
  - `VSMsgID`
    - NachrichtenID; bekommt jede Originalnachricht
  - `VSTotalOrderMsgType`
    - Typ der Nachricht, bisher: `REROUTING` („Ein Knoten an Leader") und `MULTICAST` („Leader an alle Knoten")
  - `VSTotalOrderHeader`
    - Header für internen Datenaustausch; enthält: Nachrichtentyp und -ID, sowie ggf. Ordnung (`ViewId` und Sequenznummer)

Hinweise zu Teilaufgabe 5.3

- Auslieferung von Nachrichten verzögern
  - Nachricht vom Leader entgegennehmen
  - Nachricht abspeichern ohne sie auszuliefern
  - ACK (neuer Nachrichtentyp) an alle Knoten versenden
- Nachrichten nach Empfang von ACKs ausliefern
  - Versenden der Nachricht erst nach Eintreffen der Empfangsbestätigungen von einer Mehrheit an Knoten
  - Trotz Verzögerung muss die totale Ordnung beachtet werden
  - ACKs können schon vor eigentlicher Nachricht eintreffen
- Zwischenspeicherung in geeigneter Datenstruktur
  - Schrittweiser Empfang einzelner Teile der vollständigen Nachricht
  - Signatur einer vollständigen Nachricht (Zertifikat): `<Nachricht-ID, Nachricht, Anzahl erhaltener ACKs, Sequenznummer>`
  - Auslieferung darf erst geschehen, sobald eine vollständige Nachricht die geforderten Kriterien (z. B. ACKs, Sequenznummer) erfüllt

Hinweise zu Teilaufgabe 5.4

- Versenden der Multicasts direkt vom Knoten selbst
  - Nachricht muss weder verpackt noch ausgepackt werden
  - Auslieferung der Nachricht ist zu verzögern bis Ordnung und ACKs vorliegen
- Herstellung der Reihenfolge
  - Leader versendet beim Eintreffen der Originalnachricht eine Ordnungsnachricht (ohne abermaliges Versenden der Originalnachricht)
- Anmerkung
  - Die Implementierung der ersten drei Teilaufgaben ist in geeigneter Weise so weit wie möglich wiederzuverwenden
  - Die naive Implementierung sollte weiterhin lauffähig bleiben

Vereinfachungen in Übungsaufgabe 5

- Änderungen der Gruppenzusammensetzung
  - Neuen Knoten fehlt die Nachrichtenhistorie
  - Leader-Wechsel muss bei totaler Ordnung berücksichtigt werden
  - Beides kann zur Verletzung von Zustellungsgarantien führen

  → Wechsel von Sichten wird nicht unterstützt
- Bereinigen von Zwischenspeichern
  - Zwischenspeicher von Nachrichten, z. B. zum Verhindern von Mehrfachauslieferungen, müssen irgendwann bereinigt werden

  → wird vernachlässigt

Allgemeine Hinweise

- Synchronisation
  - Es ist davon auszugehen, dass auf Instanzen der Protokollklassen von mehreren Threads parallel zugegriffen wird
- Exceptions
  - Auftretende Exceptions sind zumindest auszugegeben
- Testen der Implementierung
  - Das Skript `distribute.sh` erstellt mehrere entfernte Prozesse, die anschließend die Testanwendung `VSTestClient` ausführen
  - Die Ergebnisse werden in Logs ausgegeben, die mittels des Skripts `checklogs.sh` überprüft werden können.
  - Vor Aufruf von `distribute.sh` muss die Datei `my_hosts` mit Hostnamen von regulär erreichbaren CIP-Pool-Rechnern gefüllt werden

> *"Screen is a window manager that allows you to handle several independent screens (UNIX ttys) on a single physical terminal; each screen has its own set of processes connected to it (...)"*
>
> Posting von Christopher Laumann <net@tub.UUCP>
> 20. März 1987, Communications and Operating Systems Research Group, TU Berlin

- Wichtige Screen-Kommandos:

`Ctrl+a c`	Erstelle neues Fenster und wechsle zu diesem
`Ctrl+a Ctrl+a`	Springe zum letzten aktiven Fenster
`Ctrl+a <num>`	Springe zu Fenster <*num*>
`Ctrl+a k`	Schließe aktuelles Fenster
`Ctrl+a \`	Schließe alle Fenster und beende Screen-Instanz

- Achtung: Die Kommandos gelten für CIP-Pool-Rechner, weichen vom Standard ab und können sich auf anderen Systemen unterscheiden.