Überblick

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

Zusammenschlüsse in verteilten Systemen

- *Gruppe*
  - **Zusammenschluss von Knoten** in einem verteilten System,
    - die in der **Anzahl begrenzt** sind,
    - zumeist miteinander **gleichberechtigt kommunizieren** und
    - **gemeinsamen (globalen) Zustand** benötigen
  - Für Replikation, virtuelle Konferenzen, Netzwerkspiele, ...
- Abgrenzung
  - Client/Server-System
    - Clients sind unabhängig voneinander
    - Kommunikation nicht gleichberechtigt
  - P2P-System
    - Peers besitzen nur partielle Sicht auf das Gesamtsystem
    - Auf sehr große Anzahlen von Knoten ausgelegt
  - ... Grenzen aber fließend

Virtual Synchrony

- Probleme
  - Kein gemeinsamer Speicher
  - Keine gemeinsame Realzeit
  - Zusammensetzung der Gruppe ist oftmals dynamisch
    - Knoten treten ihr bei oder verlassen sie
    - Ausfall von Knoten
    - Abbruch von Verbindungen
- Lösungsansatz *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

Gruppenkommunikation

- Verwaltung von Gruppenmitgliedschaften und Bereitstellung von **Virtual Synchrony**
- **Austausch von Nachrichten** zwischen Knoten einer Gruppe
  - Nachrichten der Anwendung oder gruppeninterne Nachrichten
  - Umsetzung von **Zustellungsgarantien**
- Mechanismen für **Zustandstransfer**
- Beispiele
  - JGroups [`http://www.jgroups.org/index.html`]
  - Spread
  - ...

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 vorher versendete Nachr. 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
- *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
- *Kausale Ordnung*
  - Jede Nachricht wird von allen korrekten Knoten vor den von ihr (potentiell) verursachten Nachrichten ausgeliefert
  - Kausale Ordnung schließt FIFO-Ordnung mit ein
- *Totale Ordnung*
  - Nachrichten werden von allen korrekten Knoten in der gleichen Reihenfolge ausgeliefert
  - Totale Ordnung ist orthogonal zu FIFO- und kausaler Ordnung

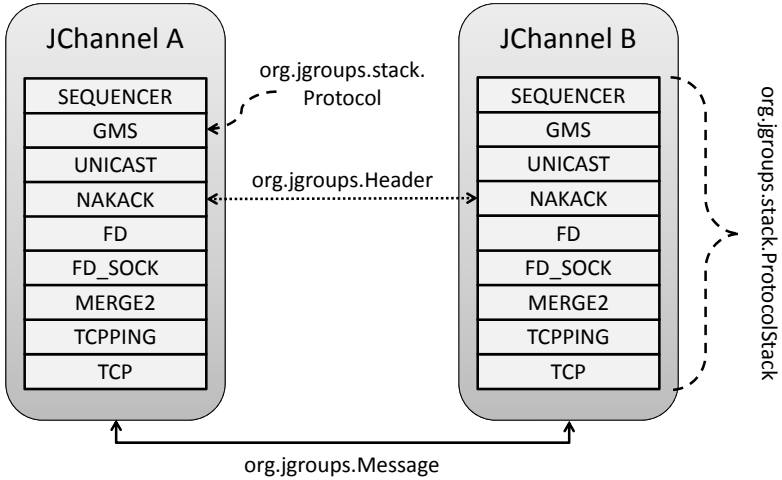
Überblick

Kurze Wiederholung zu JGroups

- **JGroups**
  - Bibliothek und **Framework für zuverlässige Gruppenkommunikation**
  - Virtual Synchrony, Zustandstransfers, verschiedene Zustellungsgarantien
  - 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`) ebenfalls asynchron (`org.jgroups.MembershipListener`)
- Siehe Folien zu Übungsaufgabe 4 (Replikation)

Protokoll-Stack

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



Protokolle

- Ausgewählte bestehende Protokolle
  - TCP/UDP
    - Transportprotokolle
  - FD
    - Heartbeat-Protokoll für Ausfallerkennung
  - NAKACK
    - Implementiert FIFO-Multicast
  - GMS
    - Protokoll für Gruppenmitgliedschaft und Sichten
  - SEQUENCER
    - Realisiert eine totale Ordnung auf Basis von NAKACK
- 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 können über *Header* interne Daten zwischen Knoten austauschen
  - Leiten von `org.jgroups.Header` ab
  - Müssen vergleichbar zu Protokollen registriert werden

```
ClassConfigurator.add(short id, java.lang.Class class);
```

  - 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 eines Header einer Nachricht oder null, wenn nicht vorhanden

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

Serialisierung

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

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

  - Klassen müssen registriert werden über XML-Datei oder zur Laufzeit (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"*
- Ansatz 1
  - 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
- Ansatz 2 (optional für 5,0 ECTS)
  - Knoten versenden Multicasts selbst
  - Leader sendet extra Nachricht mit Ordnung
  - Nachrichten werden erst ausgeliefert, wenn Ordnung bekannt
- Beide Ansätze verwenden ACKs für uniformen Multicast

Hinweise 1. Teilaufgabe

- 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
    - Liste der Mitglieder kann leer sein!
- Klasse `VSTotalOrder` (im Pub-Verzeichnis) soll als Grundlage dienen

Hinweise 2. Teilaufgabe (1/2)

- Umleiten aller Multicasts zu Leader (Ansatz 1)
  - Überprüfen, ob zu versendende Nachricht Multicast ist

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

    ansonsten Ereignis an untere Schicht weiterreichen `super.down(...)`
  - Initialisierung des Versenders der Nachricht mit lokaler Adresse, falls nicht anders festgelegt `msg.getSrc()` und `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 2. Teilaufgabe (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: 'Ein Knoten an Leader' (`REROUTING`) und 'Leader an alle Knoten' (`MULTICAST`)
  - `VSTotalOrderHeader`
    - Header für interen Datenaustausch; enthält: Nachrichtentyp und -ID, sowie ggf. Ordnung (`ViewId` und Sequenznummer)

Hinweise 3. Teilaufgabe

- 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
  - Auslieferung von Nachrichten nach Eintreffen der Empfangsbestätigungen aller Knoten
  - Trotz Verzögerung muss die totale Ordnung beachtet werden
  - ACKs können schon vor eigentlicher Nachricht eintreffen

Hinweise 4. Teilaufgabe

- Versenden der Multicasts direkt von 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 Implementierung von Ansatz 1 sollte trotz Wiederverwendung lauffähig bleiben

Vereinfachungen in Übungsaufgabe 5

- Anzahl der Bestätigungen bei uniformen Multicast
  - ACKs von einer Mehrheit der Knoten würde ausreichen
  - Führt zu einigen Sonderfällen

  → Warten auf ACKs von allen Knoten
- Änderungen der Gruppenzusammensetzung
  - Neuen Knoten fehlt die Nachrichtenhistorie
  - Leaderwechsel 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
- Ausnahmen
  - Auftretende Ausnahmen 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