JavaSpaces/TSpaces

Ausarbeitung im Rahmen der Vorlesung Verteilte Systeme im 6. Semester

Fachbereich Angewandte Informatik – Telekommunikation

Fachhochschule Bonn Rhein Sieg

vorgelegt von

Daniel Schmidt, Daniel Post

Fachhochschule
Bonn-Rhein-Sieg

Bonn im Juli 2001

Inhaltsverzeichnis

Vorteil und Herausforderung von verteilten Systemen

Systeme mit gemeinsamen Speicher

Mehrprozessorsysteme mit gemeinsamen Speicher (shared memory) greifen über einen Bus auf denselben Speicher zu. Prozesse können Parallelität in solch einem System mit Hilfe von threads ausnutzen. Ein thread kann auf den gesamten Adressbereich des Prozesses, zu dem er gehört zugreifen.

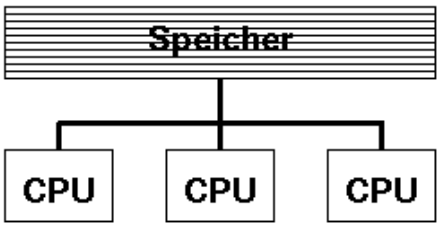

Abb.1 Mehrprozessorsystem mit gemeinsamen Speicher

Ein solches nicht verteiltes Mehrprozessorsystem ist jedoch nicht unbegrenzt skalierbar. Ein vernetztes System von mehreren Mehrprozessorsystemen gehört zu den Systemen mit verteiltem Speicher. Gemeinsame Daten werden durch explizite Kommunikation ausgetauscht. Dafür müssen Prozesszustände miteinander koordiniert werden. Eine Möglichkeit dieser Koordination ist die barrier synchronisation. Dabei rechnen mehrere Prozesse unabhängig voneinander bis zu einem bestimmten Punkt, an dem sie aufeinander warten.

Vorteile von Systemen mit verteiltem Speicher

Viele Rechenprobleme lassen sich in verschiedene kleinere Probleme aufteilen. Somit kann das Problem von mehreren Rechensystemen gemeinsam gelöst werden, wobei die Rechensysteme an den verschiedensten Standorten stehen können. Wenn die Kommunikation zwischen den verschiedenen Rechnern bzw. der Zugriff auf den verteilten Speicher gut gelöst ist, wird das Problem insgesamt schneller gelöst werden, als durch einen einzelnen Rechner auch wenn dieser eine größere reine Nennrechenleistung inne hat, als die verteilten Rechensysteme zusammengenommen.

Ein weiterer großer Vorteil ist die einfache und kostengünstige Erweiterbarkeit eines verteilten Rechensystems. Im Normalfall funktioniert eine Anwendung auf einem verteilten Rechensystem ohne Anpassung oder Veränderung, wenn eine weitere Rechenmaschine in das System integriert wird.

Ein verteiltes System kann einem Nutzer teure Ressourcen verfügbar machen. Die Ressource „Rechenleistung eines Supercomputers“ kann so z.B. für mehrere Nutzern verfügbar gemacht werden, vergleichbar einem Drucker in einem Netzwerk, und die verteilte Anwendung kann auf diese Ressource aufbauen. Dieses Ressourcensharing wirkt sich auch in der Fehlertoleranz einer verteilten Anwendung aus. So kann möglicherweise eine bestimmte Menge von Fehlern oder Ausfällen toleriert werden, da mehrere voneinander unabhängige Prozesse existieren, oder ein Prozess für einen nicht mehr vorhandenen einspringen kann.

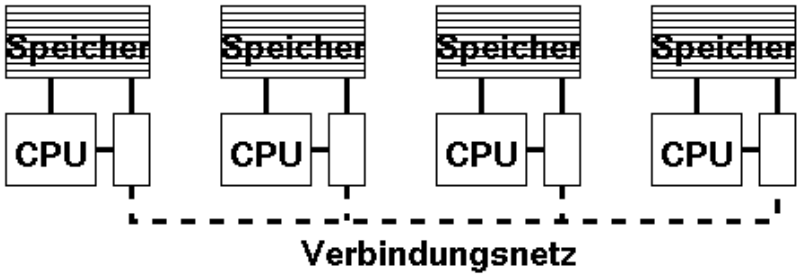

Abb.2 Mehrprozessorsystem mit verteiltem Speicher

Die Verteilung einer Anwendung kann ebenfalls aus Gründen des Software-Engineerings vorteilhaft sein. Nicht nur der Erfolg der objektorientierten Softwarekonzeption zeigen, das viele Problemlösungen oft die Dynamik einer Organisation wiederspiegeln, in der viele Prozesse asynchron arbeiten, und koordiniert werden müssen, und ein Problem zumeist nicht mehr durch eine sequentielle Abfolge von Schritten gelöst werden kann. Es erweist sich oft als eleganter, ein Problem in viele kleine Probleme aufzuteilen, wobei jedes Teilproblem durch einen Spezialisten effizienter gelöst werden kann, als durch die sprichwörtliche „eierlegende Wollmilchsau“.

Herausforderungen an Systeme mit verteiltem Speicher

Mit den Vorteilen kommen auch Schwierigkeiten bei verteilten Anwendungen, die bei einer Standalone Anwendung nicht auftreten. Eines der größten Probleme verteilter Anwendungen ist die Heterogenität der existierenden Hardware-plattformen.

Entfernte Prozesse müssen kommunizieren um zusammen arbeiten können. Diese Kommunikation geschieht über potentiell langsame Netzwerkverbindungen. Die Kommunikation kann im Verhältnis zur Prozessorgeschwindigkeit eine lange Zeit in Anspruch nehmen. Diese Zeitspanne wird als Latenz (latency) bezeichnet. Die Latenzzeit ist um mehrere

Größenordnung höher als die Kommunikation zwischen lokalen Prozessen. Dieser Tatsache muß beim Design einer verteilten Anwendung Rechnung getragen werden.

Die Latenzzeit spielt auch bei einem weiteren Problem verteilter Anwendung eine Rolle, der Synchronisation. Prozesse in einer verteilten Anwendung müssen Ihre Handlungen auf einander abstimmen. Eine Möglichkeit um die Phasen eines Algorithmus zu synchronisieren, die barrier synchronisation, wurde schon erwähnt. Genauso muss der Zugriff auf gemeinsamen Speicher synchronisiert werden.

Das Abfedern von Fehlern durch z.B. Systemabstürzen ist ein weiteres Problem in einem verteilten System. Während der Ausfall eines Prozesses in einem Standalone-System meist unweigerlich zum Absturz des kompletten Systems führt, verlangen verteilte Systeme oft, das der Ausfall eines Teils des Systems abgefangen werden kann. Für den Zugriff mehrerer Prozesse auf gemeinsamen Speicher bedeutet diese Forderung zuerst, das die Konsistenz der Daten auch bei Ausfall eines Prozesses gewährleistet wird.

Die Lösung der aufgezeigten Probleme eines Systems mit verteiltem Speicher nehmen oft einen Großteil der Ressourcen eines solchen Softwareprojekts ein. JavaSpaces von Sun und Tspaces von IBM sind zwei Ansätze, um den Softwareentwicklern die Vorteile eines verteilten Speichers anzubieten, und dabei den Entwicklern die Bürde abnimmt, die entstehenden Nachteile auszugleichen. Beide Systeme basieren auf dem Linda-System, das Anfang der 80er in Yale entwickelt wurde.

Tuplespaces in Linda

Grundlage von Linda ist der Tupelraum (Tupelspace). In diesen können Wertetupel abgelegt und wieder entnommen werden. Der Tupelraum ist eine Abstraktion von gemeinsamen Speicher, dessen Implementierung nicht festgelegt ist.

Aus dem Tupelraum kann ein Tupel entnommen werden, und natürlich ein Tupel in den Tupelraum abgelegt werden. Beim Entnehmen-Operation gibt es eine blockierende und nicht-blockierende Variante. Die blockierende Variante wartet solange bis ein gewünschtes Tupel im Tupelraum vorhanden ist und somit entnommen werden kann. Auch das einfache Lesen eines Tupels, ohne dieses zu entnehmen ist natürlich möglich.

Man kann sich den Tupelraum als eine Art Datenbank vorstellen, in die ein Gerät, Programm oder Prozess seine Daten schreibt, und persistent speichern kann.

Die Suche nach einem bestimmten Tupel beim Lesen oder Entnehmen geschieht assoziativ über die Angabe eines Musters. Die Tupel werden nicht an einen bestimmten Adressplatz geschrieben, sondern über ihren Inhalt adressiert. Das Muster gibt dabei an, welchen Inhalt das gesuchte Tupel haben soll.

Mit der Kommunikation über einen Tupelraum lassen sich z.B. „Producer-Consumer“-Beziehungen erstellen, bzw. „Auftraggeber-Auftragnehmer-Beziehungen abbilden. Der Tupelraum dient dabei als Auftragspuffer. So kann ein Auftragstupel folgendermassen aussehen: (berechne, 42,933). Ein Auftraggeber würde durch die Operation out(berechne, 42,933) den Auftrag in den Tupelraum schreiben. Ein Auftragnehmer könnte diesen Auftrag durch die Operation in(berechne,*,*) erhalten. Die * dienen dabei als Platzhalterzeichen.

Die Kommunikation über den Tupelraum ist asynchron, anonym und persistent. Die Asynchronität rührt daher, das die Kommunikationspartner nicht zu einem bestimmten Zeitpunkt miteinander kommunizieren müssen. Ebenfalls muß keiner der Kommpartner wissen, wer der jeweils andere ist. Sie müssen lediglich beide über die Existenz des Tupelraums bescheid wissen. Die Daten im Tupelraum können länger leben als die Programme die sie erzeugt haben. Somit sind die Tupel im Tupelraum persistent.

Es existiert eine Implementierung von Linda namens Linda-C. Diese erlaubt nur einfache und zusammengesetzte Datentypen für die Tupel in dem Tupelraum, und ist per se nicht objektorientiert. Aus diesem Grund haben Sun und IBM eigene Versionen dieser Technik entwickelt. Die beiden Ansätze unterscheiden sich jedoch in ihren Schnittstellen und sind daher nicht miteinander kompatibel.

Im folgenden wird die Implementierung eines Tupelraums von Sun, JavaSpaces, vorgestellt. Die Implementierung von IBM wird anschließend nur noch vergleichend erläutert.

JavaSpaces

Installation von JavaSpaces

JavaSpaces ist in Java geschrieben und ein fester Bestandteil von Jini. Das JavaSpaces Technology Kit (JSTK) liegt zur Zeit in der Version 1.0 vor, und kann auf der Java Developers Connection web site heruntergeladen werden. Im JSTK ist der Quellcode und der Binärcode der JavaSpaces Software enthalten, sowie die API-Dokumentation. Damit JavaSpaces funktioniert, muss vorher Jini lauffähig installiert worden sein.

Die Entry-Objekte

Pendant zu den Tuplen bei Linda sind die Entry-Objekte bei JavaSpaces. Ein Objekt kann als Entry-Objekt einem JavaSpace hinzugefügt werden, indem seine Klasse das das Entry-Interface implementiert. Es wird lediglich die Deklaration des Interface gefordert, das Interface fordert nicht die Implementierung irgendeiner Methode. Clients können Entry-Objekte aus dem JavaSpace entnehmen, lesen oder hineinschreiben.

Der JavaSpace

Der Tupelraum, bei JavaSpaces schlicht Space genannt, wird zentral von einem Server verwaltet. Auf diesen kann ein client über ein Netzwerk zugreifen, und dessen Schnittstelle entfernt nutzen, indem wie bei RMI ein ClientStub erzeugt wird, der dem Client die Schnittstelle lokal anbietet, wodurch die Kommunikation transparent gehalten wird. Die Methoden der Schnittstelle entsprechen in ihrer Funktionalität denen von Linda: write(), read() und take(). Read und Take sind dabei blockierende Methoden. Die Wartezeit kann durch Angabe von Timeout-Werten begrenzt werden.

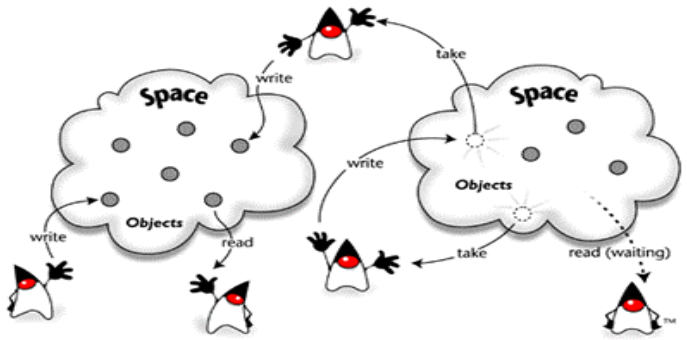

Abb. 3 Prozesse benutzen für Ihre Koordination Spaces und einfache Operatione

Auf die Anfrage eines Clients liefert der Space Server einen passenden Eintrag zurück. Bei mehreren passenden Einträgen wird ein beliebiger Eintrag zurückgegeben. Dies kann sich als Problem herausstellen, wenn z.B. alle passenden Entries zurückgegeben werden sollen.

Der Benachrichtigungsmechanismus

Ein Client kann sich mit der notify-Methode bei einem Space anmelden und sein Interesse an bestimmten, neu eingehenden Schreibereignissen kund tun. Trifft ein Eintrag ein, der dem angegebenen Template entspricht, wird ein Eventhandler aufgerufen. Dieser nimmt das Ereignis auf, und ergreift die geeigneten Maßnahmen.

Transaktionssicherheit

Dem Problem möglicher Inkonsistenzen durch den Ausfall eines Clients ist bei JavaSpaces Rechnung getragen durch die ‚durch Jini Technik gegebene, Möglichkeit Methoden transaktional zu behandeln. Kann zum Beispiel ein Entry durch einen Client entnommen und verändert werden, ist aber durch die Anwendung verlangt, dass ein Entry wieder zurückgeschrieben werden muß, so könnte ein Ausfall des Clients nach Entnahme eines Entries zu Inkonsistenzen führen. Die Zusammenfassung der atomaren Take- und Write-Operationen zu einer Transaktion bewirkt daher, dass in jedem Fall beide Teiloperationen ausgeführt werden. Kann eine der beiden Operationen nicht ausgeführt werden, so wird keine Operation ausgeführt. Bei einem Ausfall des Clients nach der Take-Operation würde also der Entry durch den Transaktionsmechanismus wieder zurück in den Space geschrieben werden.

Unterschied zu einer Datenbank

Während eine Datenbank einen großen Overhead an Programmierung, Formular Design, Type Matching und Indexierung benötigt um Ergebnisse zu erhalten, bietet JavaSpaces ein locker verbundenes, leicht zugängliches Repository von Informationen bzw. Objekten. Die Identität eines Clients oder Servers ist im Gegensatz zu einer ausgewachsenen Datenbank nicht relevant.

Dadurch das ein JavaSpace Objekte sowohl nach Typ als auch nach Werteinhalten finden, vergleichen und referenzieren kann, bietet es jedem objekt-basierten Programm bzw. Client die Möglichkeit sich in das System einzuklinken. Dies bietet die Möglichkeit die verschiedensten Funktionen und Prozesse in einer Netzwerkumgebung zu koordinieren.

Die HelloWorld Beispiel-Anwendung

Wir wollen in diesem Abschnitt die Entwicklung einer einfachen verteilten HelloWorld-Anwendung mit Hilfe von JavaSpaces erläutern. Das Beispiel ist entnommen aus dem Buch „JavaSpaces – Principles, Patterns and Practice“.

Wir wollen in unserem Space Messages als Entrys speichern. Die Klasse Message muß ein Feld implementieren, in dem die Nachricht gespeichert werden kann. Damit es als Entry in einen Space geschrieben werden kann, muß es das Interface Entry implementieren:

```
public class Message implements Entry {
      public String content;

      public Message() {
      }
}
```

Ein Message Entry kann nun instantiiert werden, und das content Feld auf „Hello World“ eingestellt werden.

```
Message msg = new Message();
msg.content = "Hello World";
```

Nun erzeugen wir die Instanz eines JavaSpaces, in die wir den Message Entry mit der Write-Methode hineinschreiben können. Die getSpace()-Methode der SpaceAccessor Klasse erzeugt die Instanz einer Klasse, die das JavaSpace-Interface implementiert.

```
JavaSpace space = SpaceAccessor.getSpace();
space.write(msg, null, Lease.FOREVER);
```

Um einen Entry in einem JavaSpace zu finden, wird eine Vorlage, ein Template als Vergleichsobjekt angegeben. Alle Felder, die das Suchobjekt mit einem bestimmte Wert gefüllt haben soll, hat das Template mit diesem Wert gefüllt. Die restlichen Felder, bei denen der Wert des Entries egal ist, werden mit dem Wert null gefüllt. Wir erzeugen ein Template, das das content-Feld mit dem Wert null gefüllt hat. Somit würde jeder Message-Eintrag in unserem Java-Space auf das Template passen.

```
Message template = new Message();
```

Nun können wir mit Hilfe des Templates und der read-Methode ein Message-Objekt aus dem JavaSpace auslesen. Wir wissen das wir ein Message-Objekt erhalten werden, und können es daher casten.

```
Message result = (Message)space.read(template, null, Long.MAX_VALUE);
```

Nachdem wir das Objekt erhalten haben, können wir das Content-Field ausschreiben, und erhalten die Nachricht “Hello World!” auf dem Bildschirm.

```
System.out.println(result.content);
```

Unterschiede zwischen JavaSpaces und TSpaces

Tspaces von IBM ist eine zweite existierenede Implementierung von Tupelräumen. Es ist ebenfalls in Java implementiert, unterscheidet sich aber gegenüber JavaSpaces in der Schnittstelle zum Tupelraum.

Während JavaSpaces beliebige Objekte durch die Implementierung des Entry-Interface in einem Space speicherbar macht, ist Tspaces ist näher an den Tupeln orientiert. Ein Tupel kann zwar ebenfalls eine beliebige Klasse sein, es muß dann aber eine Unterklasse der Subclassable Tuple Klasse sein. Es existiert in TSpaces auch eine reine Tupel Klasse. Die Objekte der Klasse Tuple werden dann einfach mit den gewünschten Feldwerten instantiiert.

Während man in JavaSpaces auf die Felder der Tupel zugreift, indem man sie entgegen objektorientierter Kapselungskonzepte als public deklariert, greift man in TSpaces durch die Zugangsmethoden get() und set() auf die Felder zu.

Es bestehen auch leichte Unterschiede in der Signatur der Methodenaufrufe. Die Instanziierung eines TupleSpace geschieht in TSpaces einfach durch den Aufruf des Konstuktors der Klasse TupleSpace. Auch die Methode write besitzt eine andere Signatur, hat aber dieselbe Funktionalität wie in JavaSpaces. Die blockierenden read und take Methoden heißen in TSpaces waitToRead bzw. waitToTake und die Methode takeIfExists in JavaSpace ist in TSpace durch die take-Methode mit abgedeckt.

Desweiteren bietet TSpaces einige weitere Methoden an, die in JavaSpaces nicht implementiert sind. Dazu gehört z.B. die scan()-Methode, mit der alle auf ein Template passenden Tupel ermittelt werden können.

Bewertung von JavaSpaces

JavaSpaces bzw. TSpaces erleichtert es dem Anwendungsentwickler, verteilte Anwendungen zu erstellen. Viele Schwierigkeiten der gemeinsamen Nutzung eines Speichers werden dem Entwickler abgenommen. Die Überschaubarkeit der Methoden machen die Einarbeitung in das Framework einfach. Der Entwickler kann sich auf das Protokoll seiner Anwendung konzentrieren, und muss sich nicht um die Kommunikation und Synchronisation der beteiligten Clients kümmern.

Jedoch ist es fraglich, wie die Performance eines Space Servers mit erhöhter Belastung und großen Anzahlen von zu verwaltenden Tupel-Objekten aussieht. Ebenso ist das Problem der Sicherheit des Space Servers ungeklärt.

Die JavaSpaces-Technologie scheint derzeit noch eine Lösung lediglich für Intranets mit bekannten sich vertrauenden Teilnehmern zu sein.