VSDB

Space Based Computing

Ausarbeitung

Daniel Dimitrijevic Thomas Traxler

16. Dezember 2013

5AHITT

Inhaltsverzeichnis

1 Erklärung

1.1 Wofür steht Space-Based Computing?

Space-Based Computing (fortan SBC) hat seine Ursprünge im parallel programming, und stellt dabei hauptsächlich ein Datenorientiertes Modell zur Koordination dar. Wie aus dem Namen eindeutig hervorgeht handelt es sich hierbei um ein Modell welches auf 'Spaces' basiert. Ein Space ist hierbei gleichbedeutend mit einem (logischen) Ort auf welchem Daten von mehreren Komponenten geteilt verwendet werden. Diese Komponenten können im einfachsten Fall 'write', 'read' und 'take' Aktionen ausführen. Ein Write steht hierfür für das zur Verfügung Stellen eines neuen Datenteil an alle anderen Komponenten in diesem Space, ein Read für das Lesen von Daten ohne diese zu entfernen und ein Take für das Lesen und entfernen einer Datei aus dem Space, auch destructive read genannt. Im einfachsten Fall hat ein Space nun das Datenmodell eines Tuples.
Eine Hauptanforderung die an einen Space hier nun gestellt wird ist das persistente Aufbewahren aller Daten die sich in ihm befinden, auch bei Systemausfällen. Wie das SBC modelliert und implementiert wird kann sehr stark variieren, je denn gegebenen Anforderungen und Wünschen entsprechend. Was hierbei vor allem variiert ist die Zahl der Geräte auf dem der Space implementiert ist und die Zahl der Geräte die auf diesen Space zugreifen, diese Zahlen sind prinzipiell, wenn nicht durch den Anwendungsfall anders umgesetzt, voneinander unabhängig und befinden sich jeweils im Bereich von 1 bis n.
Das SBC-System stellt hierbei eine logische zentrale Einheit dar welche nicht spezifiziert wo sie genau überall Physikalisch vorhanden ist. Der Ort wo eine Datei schlussendlich wirklich physikalisch abgespeichert wird kann dabei nach verschiedensten Methoden ausgewählt werden, in der simpelsten Form wird das selbe Prinzip wie bei Tuples angewendet oder es werden andere Prinzipien verwendet wie FIFO, LIFO, keys, geo-coordinates oder noch kompliziertere, auch das ist vom gegebenen Anwendungsfall abhängig. [**?**]

1.1.1 Unterschied zwischen SBC und Cloudcomputing

Um nun der Verwechslung von SBC und Cloudcomputing vor zu Beugen sie hier nun gesagt, zu aller erst behandelt SBC lediglich die Daten und Cloudcomputing stellt noch viele weitere Ressourcen (wie zB. Rechenleistung) zur Verfügung. Auch zu Cloud-Storrage kann man abgrenzen, Cloud-Storrage behandelt nicht nur die interne Abspeicherung der Daten sondern auch vor allem wie diese dem Benutzer zur Verfügung gestellt und abstrahiert werden, bei SBC geht es hauptsächlich um die interene Verteilung und Kommunikation der Komponenten untereinander und stellt dabei einen Datenkanal zwischen

den Komponenten des Spaces dar beziehungsweise zur Verfügung.

1.2 Einsatzbereiche

SBC wird vor allem immer dort verwendet wo Daten verteilt abgespeichert werden sollen. Dies kann auf verschiedenste Arte durchgeführt werden um so 'Bottlenecks' zu schließen. SBC bietet durch die Verteilung des Spaces auf mehrere Komponenten grundlegend immer mehr zur Verfügung stehenden Speicherplatz, wodurch man je nach Implementierung verschiedene Vorteile gewinnen kann. Man kann diesen Speicherplatz beispielsweise direkt logisch zur Verfügung stellen und so den logisch zur Verfügung stehenden Speicherplatz erhöhen, man kann die notwendige Kommunikation auf einer Komponente verringern (durch Aufteilung) und man kann für Redundanz sorgen. Die Vorteile lassen sich hierbei auch annähernd beliebig Kombinieren wobei zu beachten ist, das 2 Kombinierte Vorteile auch bei bester Umsetzung so gut wie immer einen dritten Vorteil verringern oder noch andere Nachteile nach sich ziehen. Redundanz schlägt sich Beispielsweise immer auf den zur Verfügung stehenden Speicherplatz nieder, was durch erhöhte Kommunikation zwischen den Komponenten jedoch wieder verringert werden kann.
Ein Beispiel für einen weiteren Vorteil wäre zum Beispiel die Möglichkeit an jeder Komponente Zugang zum ganzen Space zu haben, was in einem Master-Client System erhöhte Kommunikation zum Master mit sich bringt, außerdem müssen die Clients in diesem Fall auch durchgehend die Möglichkeit haben zumindest einen Master ansprechen zu können. Dieser Vorteil stellt sich automatisch ein wenn man auf ein Peer-to-Peer System wechselt, wobei auch oftmals stark belastete Master-Komponenten nicht mehr notwendig sind aber in der Regel wieder mehr Kommunikation zwischen den Komponenten notwendig ist.

2 Grundlegende Prinzipien

2.1 Space-Based Computing Paradigmen

2.1.1 Tuple Spaces

2.2 Mapping

2.3 EAI

3 Im genaueren betrachtet

3.1 Tuple Spaces

Tuple Spaces(Linda)

Ein Tupel Raum ist eine Ausführung des assoziativen Gedächtnisses Modell für verteilte / paralleles computing. Es bietet eine Bibliothek von Tupeln, auf die gleichzeitig zugegriffen werden kann. Tupel sind Begriffe, mit null oder mehr Argumente und einem Schlüssel.

Die Sammlung von Tupeln unterstützt einige Grundfunktionen, wie zB das Hinzufügen eines Tupels in den Raum (Schreiben) und Entfernen eines Tupels aus dem Raum (nehmen). Das Tupel Sammlung wird von einem Netzwerk über meherer Server aufbewahrt und gemanaged. Mehrere Threads können zur selben Zeit auf den selben Raum zugreifen. Nun gehen wir einwenig auf Linda ein.

Ziel von Linda ist es, Prozessen einer Anwendung zu erlauben, miteinander zu kommunizieren, ohne identifikations Informationen des Datensatzes zu wissen. Früher ging Linda von einem Tupelraum als einer abstrakten Umgebung aus. Verschiedene nebenläufige Prozesse eines verteilten Programms kommunizieren über einen gemeinsamen Tupelraum dadurch, dass jeder dieser Prozesse diesem Tupelraum beliebig Tupel hinzufügen und Tupel daraus entfernen kann.

Ein Prozess A lässt einem Prozess B Information zukommen, indem er eine gebündelte Menge von Werten, ein Tupel, im Tupelraum ablegt. Prozess B kann anschließend das Tupel lesen oder es aus dem Tupelraum entfernen, womit der Kommunikationsakt abgeschlossen ist. Der Prozess A benötigt weder einen Namen, eine Adresse oder sonstige identifizierende Information von Prozess B; für Prozess A ist es völlig irrelevant, ob Prozess B oder irgendein beliebiger anderer Prozess, mehrere Prozesse oder kein Prozess das Tupel liest. Derjenige Prozess, der das Tupel des Prozesses A entnimmt, muss nicht einmal zur gleichen Zeit wie Prozess A aktiv sein oder existieren. Das von A generierte Tupel ist von seinem Erzeugerprozess vollkommen unabhängig, das bedeutet, dass Prozess A zum Zeitpunkt der Entnahme seines Tupels durch Prozess B schon lange beendet sein kann.

Die hieraus resultierende zeitliche und aufgrund der Verteilung auch räumliche Entkoppelung ermöglicht einen Entwurf verteilter Protokolle, die flexibel und robust auf die Herausforderungen verteilter Programmierung durch Latenz, erhöhten Synchronisationsaufwand und mögliche Teilausfälle des Systems reagieren können.

3.2 Triple Spaces

Wie Tuple Spaces will auch Triple Spaces auf parallelen Verarbeitung von Anfragen auf Daten. Es übernimmt das Virtuel shared Memory

Zusammensetztung aus: Tuple Spaces Semantic Web Web Service

Semantic Web Technologie bietet Vorteile bei der Verwaltung und Anpassung von Kommunikation durch semantische Annotationen. Durch die Erstellung von Ontologien bieten sie Konsens Terminologien und erleichtern interoperability. „Matchmaking“ von semantischtisch verknüpften Daten kann die Notwendigkeit von direkten Zuordnungen von Kommunikationsbedarf reduzieren.

Web-Service Technologie bietet eine "Virtuelle Komponenten-Modellßur vereinheinfachung der heterogenenen Welt von Komponenten. Es erlaubt es, bestehende Funktionalität zu nutzen ohne die Last der Mittelware spezifischen Eigenheiten, wie die invokation-Mechanismus, Transport-Protokoll usw. Web Services sind eine gut verstandenes comkommunikationsstruktur und-Architektur für Unternehmensanwendungen. Letztlich Web Dienstleistungen sind ein weiterer großer Schritt in Richtung Lösung des EAI Problem.

3.3 Reliable Message

Reliable Message verhält sich wie Tuple Space computing. Nur das man hier nicht von Tuples spricht sondern von Messages. Die Reliable Message Technologie geht von mehreren Channels aus die die geteilten Daten beinhalten und jeder dieser Channel kann von einem einem oder mehreren Server „unterstützt“ werde d.H. das die geteilten Daten auf einem oder mehreren Servern liegen und damit abgesichert werden können.

3.4 Peer-to-Peer Architekturen

Peer-to-Peer (fortan p2p) Architekturen bieten nun ganz eigene Möglichkeiten aber auch Anforderungen an und für ein SBC System. P2p bedeutet Gleichheit aller Komponenten, mindestens insofern, als dass es keine bestimmenden und steuernden (Master) Komponenten gibt, alle Komponenten gleichgestellt sind. Wobei zu beachten ist, dass es nicht nur erlaubt, sonder erwünscht ist, wenn Unterschiede bezüglich zB. den Komponenten zur Verfügung stehenden Ressourcen berücksichtigt werden.

3.4.1 Weshalb p2p bei SBC?

Ein (gutes) p2p System bietet vor allem die Möglichkeit einfach Komponenten hinzu zu fügen, heraus zu nehmen oder aus zu tauschen, was gerade für einen Space der nicht vollständig definiert sein muss einen guten Vorteil bieten kann. Durch eine p2p Lösung kann sehr oft hohe Flexibilität des Systems erreicht werden, hier sei nun gleich angemerkt, sollten die Anforderungen möglichst effiziente Nutzung von begrenzten oder klar definierten Ressourcen sein, so ist dies zu berücksichtigen und ein p2p Ansatz sehr wahrscheinlich nicht die beste Lösung.

Ein p2p System liefert einem außerdem den Vorteil, dass man bei sehr dynamischen Systemen nicht oder nur bedingt darauf achten muss, dass alle Komponenten Arten (zB. Master-Servant) in einem effizienten Gleichgewicht zueinander stehen da Prinzipiell alle Komponenten der selben Art entsprechen sollten.

3.4.2 P2p Arten/Routing

Die Arten von p2p Systemen kann man in der Regel dadurch unterscheiden wie das Routing innerhalb des Systems funktioniert, d.h. wie schafft es Komponente A mit Komponente B-Z Kontakt auf zu nehmen. Das Routing in (größeren) p2p Systemen ist entscheidend dafür wie skalierbar, effizient, sicher und ressourcenschonend es ist.

Prinzipiell kann man p2p Systeme in Unstrukturierte und Strukturierte Systeme unterteilen. Unstrukturierte Systeme stellen dabei den weit bekannteren Teil der Systeme und umfassen unter anderem Zentralisierte Systeme (zB. Napster), Pure p2p Systeme (zB. Gnutella 0.4) oder Hybride p2p Systeme (zB. Gnutela 0.6) wärend Strukturierte p2p Systeme beispielsweise mit Distributed-Hash-Tables realisiert werden können. [1]

3.4.3 Unstrukturierte p2p Systeme

Zentralisiertes p2p

Zentralisiertes p2p ist nur bedingt ein tatsächliches p2p System. Es besteht zwar durchaus aus vielen Gleichberechtigten Komponenten (zb. Datenspeichereinheiten) welche die Kommunikation und Organisation selbstständig regeln, jedoch existiert hierbei ein zentraler Server bei welchem sich die Peers registrieren müssen und welcher den Peers die Information mitteilt wie diese den von ihnen gesuchten Peer erreichen können. Dies führt zwar einerseits zu geringerem Traffic zwischen den Peers da diese lediglich mit der zentralen Einheit kommunizieren müssen um einen anderen Peer zu erreichen, außerdem müssen die Peers lediglich abspeichern wie sie den zentralen Server erreichen, jedoch entspricht es nur bedingt tatsächlich dem p2p Konzept.
Die Sicherheit des Systems hängt hierbei direkt von der der Sicherheit der Zentralen Einheit ab.

Pures p2p

Ein pures p2p System besteht aus x gleichberechtigten Komponenten die eine Verbindung mit y (y<x) Komponenten besitzen. Hier existiert keine zentrale Einheit und jede Komponente versucht möglichst direkt Kontakt mit jeder anderen Einheit auf zu nehmen. Wie das Routing innerhalb des Systems durchgeführt wird kann sehr Unterschiedlich sein, jedoch ist es in der Regel bis zu einem gewissen Grad 'chaotisch' da es auf sehr kurzfristige Änderungen reagieren können muss. Pures p2p ist in der Regel sehr dynamisch da jederzeit neue Komponenten an jedem Ort hinzu kommen können

[1] [?]

und diese Verbindungen zu allen Möglichen anderen Komponenten ziehen können, bzw. permanent abspeichern wie diese erreichbar sind. Da dies jedoch meist (pseudo-)zufällig vonstatten geht ist Effizienz sehr oft nicht gegeben da zB. geographische Verteilung oder Belastbarkeit der Knoten nicht berücksichtigt wird. Das hingegen kann dazu führen dass man eine Komponente erreichen will die im Nebenraum platziert ist aber aufgrund des chaotischen Routings und der Chaotischen Verbindungsfindung dafür über 3 Kontinente gerouted wird.

Hybrid p2p

Hybrid p2p ist eine Kombination aus zentralisiertem und purem p2p. Bei hybridem p2p bilden die Komponenten des Systems 'Gruppen' nach bestimmten Faktoren (meistens geographische Verteilung) und bestimmen eine oder mehrere Hauptkomponenten welche den Großteil des Routings zu den anderen Gruppen übernimmt. Die bestimmte Hauptkomponente ist in der Regel die welche am meisten Netzwerklast verarbeiten kann und am meisten Kontaktdaten zu anderen Komponenten abspeichern muss. Diese lokale Hauptkomponente ist dadurch in der Ausübung ihrer eigentlichen Tätigkeit eingeschränkt, jedoch erhält das gesamte Netzwerk so eine Struktur welche den Nachteilen der Chaotischen Verbindungsführung entgegenwirkt und es kann dennoch leicht dynamisch gehalten werden da bei Ausfall einer Hauptkomponente einfach eine neue bestimmt werden kann. Auch kann man hier Gruppen wieder zu Gruppen zusammenfassen. (zB. Gruppenebene 1 fasst alle Komponenten eines Landes zusammen, Gruppenebene fasst alle Gruppen der ersten Ebene eines Kontinents zusammen)
Hybrid p2p Systeme werden auch als 2. Generation unstrukturierter p2p Systeme bezeichnet.

3.4.4 Distributed Hash Tables

Distributed Hast Tables (fortan DHT) stellen den hier angeführten Vertreter von strukturierten p2p Systemen dar. Bei einem strukturierten System ist zu jederzeit eine ganz bestimmte Komponente für eine (bzw. mehrere) ganz bestimmte Anfrage zuständig. und zwar insofern als dass für jede theoretisch mögliche Anfrage innerhalb des Systems eine Komponente zuständig ist, nicht mehr und nicht weniger.
Um diesen recht abstrakten Satz nun ein wenig zu veranschaulichen, nehmen wir an ziel des Systems ist es eine Datenbank auf zu bauen. So hat jeder Datensatz einen eindeutigen ihm zugehörenden Hashwert, für diesen Hashwert ist nun eine spezielle Komponente zuständig und diese muss wissen wo dieser Datensatz genau zu finden ist.
Ein konkreteres Beispiel: Die Datensätze bekommen einen Hashwert im Bereich 1 bis 9 und wir haben 3 Komponenten. Jede Komponente ist nun für einen Bereich der Hashwerte zuständig, sprich Komponente 1 muss wissen wo sich datensätze 1 bis 3 befinden und welche Komponente sie benachrichtigen kann um Datensätze 4-9 zu finden.

Chord Algorithmus

4 Namhafte Implementierungen

4.1 JavaSpaces

JavaSpaces ist eine Spezifikation des Konzepts Object Spaces in der Programmiersprache Java. Ein Object Space ist hierbei ein assoziativer Speicher von verteilten, über das Netz erreichbaren Objekten. Kommunikationspartner (peers) kommunizieren ausschließlich indirekt über diese Objekte (stateful communication and coordination). Dadurch etabliert der JavaSpace einen äktiven, verteilten Datenraum", wie er in keiner anderen Technologie geschaffen wird (traditionelles Grid-Computing). Einige Ansätze der Jini-Technologie kommen hierbei zur Anwendung. Bei der Idee, die sich hinter den JavaSpaces verbirgt, handelt es sich nicht um eine revolutionäre Neuerung, sondern sie basiert im Wesentlichen auf den Linda TupelSpaces.

Die Gründe, warum JavaSpaces eingesetzt werden, sind vielfältig. Meist wird Skalierbarkeit und Verfügbarkeit bei gleichzeitiger Reduzierung der Gesamtkomplexität angestrebt.

4.2 Corso

Corso(Co-ORdinated Shared Objects) basiert auf Forschungsarbeiten der Technischen Uni Wien. Software Entwicklung in heterogenen verteilten Systemen bringt immer zusätzlich Komplexität. • Lokatoin: Addressierung und Lokalisierung von Ressourcen • Replikation: Verwalten von Daten Kopien an dislozierten Orten • Transaktion: Synchronisationsmechanismen konkurrierender Zugriffe auf verteilte Ressourcen • Skalierbarjeit: Die Möglichkeit zur transparenten Erweiterung des Systems im Zuge von wachsenden Anforderungen und Belastung an das Gesamtsystem • Lastverteilung: Die faire und gleichmäßige Verteilung von Aufgaben an verteilte Komponenten • Fehlersicherheit: Die kompensation aufallender Rechner und Persistzenz von flüchtigen Daten Corso ist eine Middleware. Corso erleichtert die Addressierung und vereinfacht das arbeiten mit Spaces.

4.2.1 Virtual shared Memory

Ein Virtuals shared Memory stellt einen konzeptionell hohne Abstraktionslevel für den Datenaustasch in verteilten Systemen dar. Es stellt einen Raum dar auf dem parallel verteilte Prozesse eine konsistente Sicht haben. Die verteilten Objekte werden für Speicherung von Informationen, Kommunikation, und Synchronisation von Prozessen verwendet.

4.3 XVSM

XVSM (eXtensible Virtual Shared Memory). XVSM ist eine Middleware technology die Daten in "Spaces" speichert und für andere Peers teilt. Dieser Weg bringt einige Vorteile wie die Daten werden auf mehreren verschiedenen Computern verteilt, damit wird die ausfallswahrscheinlichkeit der Daten reduziert.

XVSM ist keine middleware sondern eine technology die implementiert und verwendet werden kann. Unterschied zum Linda(javaSpaces) ansatz ist, dass die Daten mit eintregen, coordinaten und containern leichter gefunden werden können.

4.4 TinySpaces

Tinyspaces setzt auf das XVSM prinzip auf und benutzt das .Net Framework. Da es bei dem XVSM Prinzip noch nicht vollkommen mit allen Implementierungen kompatibel war erschuf man Tinyspaces. TinySpaces soll es erleichtern mit space based computing zu arbeiten. Da es unnötige schichten entfern. TinySpaces nutzt die unterstützung von Contracts die sagen wie man ein bestimmtes Interface ansprechen soll. Durch die Nutzung von Contracts ist es möglich die angesprochenen Componenten zu ändern. Da man bei der programmierung der Aplication darauf zielen sollte das Alle Componenten auf denn Contract aufbauen sollten. TinySpaces setzt auf CAPI-1 auf damit sie denn wechsel von Componenten während der Laufzeit zu ändern.

4.5 GSpaces

Geht von dem selben Ansatz aus wie JavaSpaces(Tuple Spaces) hat aber einen großen Unterschied. GSpaces hat zwar Tupel aber benutzt diese auch mit Replikationsrichtlinien. Bei einem Aufruf wird aber ein lokaler Aufrufshändler aufgerufen und dann wird nachgeschaut welche Operation ausfgeführt werden muss. Bei einem aufruf wird erst mal nachgeschaut welche Richtlinien angewendet werden müssen. Nachdem man die Auswahl getroffen hat wird die Anfrage weiter an einen Verteilungsmanager weitergeleitet. Wenn dann zB gelesen werden soll und es auf Master-Slave-Richtlinie herrscht wird dann einfach die Read Operation auf dem lokalem Datensatz(Slice) gelesen. Falls es aber eine write Operation ist muss der Verteilungsmanager beim Master Server anfragen ob er schreiben darf und dann kann er erst weiter schreiben.

5 Conclusio

6 Quellen