TECHNISCHE UNIVERSITÄT ILMENAU
Fakultät für Informatik und Automatisierung
Institut Telematik
Fachgebiet Verteilte Anwendungen

P2P-Architektur zur verteilten Berechnung evolutionärer Algorithmen

Zusammenfassung

Die Peer-2-Peer Technologie wird bisher vorwiegend mit Daten-Tauschbörsen im Internet in Verbindung gebracht, dabei ist sie auch für verteilte Berechnungen interessant. Die Arbeit untersucht Möglichkeiten zur Nutzung der Rechenleistung verschiedener Rechner, die über ein P2P-Netz verbunden sind. Insbesondere werden Aspekte des Verteilens von Berechnungen betrachtet. Des öfteren wird dabei der Bezug zu einer Bibliothek zur Entwicklung evolutionärer Algorithmen hergestellt. Sie dient als Anwendungsfall zur Erforschung und Entwicklung von Ansätzen zur verteilten Berechnung.

Michael Rasenberger
Matrikelnr.: 29354

Inhaltsverzeichnis

1 Netzwerke

Computernetzwerke sind heutzutage aus der Informationsverarbeitung nicht mehr wegzudenken. Viele Anwendungen nutzen die Vorteile, die durch die Vernetzung von Rechnern entstehen. Ebenso macht es eine Menge von Problemen nötig, netzwerkbasierte Lösungen zu entwickeln. Der Ressourcenbedarf einer Vielzahl heutiger Aufgaben ist dermaßen hoch, dass die Erbringung durch einzelne Rechner ineffektiv oder unmöglich ist.

Mit der Entdeckung des Internets durch die "breite Masse" wurde diese Entwicklung noch verstärkt. Die Nutzung bestehender Anwendungen durch viele Nutzer setzt Netzwerke voraus, welche den Zugriff auf die Anwendungen von verschiedenen Orten aus ermöglichen.

Eine Konsequenz aus den steigenden Ressourcenanforderungen heutiger Anwendungen ist die Aufteilung auf verschiedene Rechner. Verteilte Anwendungen koordinieren diese Verteilung mit dem Ziel eine Gesamtaufgabe zu lösen. Unter Ressourcen werden zum Beispiel die benötigten Datenbestände und die Rechenleistung verstanden.

Bestehende (und auch zukünftige) Rechnerverbunde weisen oftmals eine heterogene Struktur auf, begründet durch die Vielfalt an Hardware und Betriebssystemen (siehe Internet).
Durch Netzwerke wird es Rechnern ermöglicht miteinander zu kommunizieren. Für nützliche Anwendungen ist es erforderlich diese Kommunikation zu strukturieren.

1.1 Client-Server-Modell

Dieses Modell für Kommunikationsprotokolle ist in der bestehenden Netzwerkwelt wohl am stärksten vertreten. Es bildet die Basis für Internetprotokolle wie z.B. HTTP und FTP.

Das Client-Server-Modell zeichnet einen bestimmten Rechner als Server und andere Rechner als Clients aus. Es handelt sich dabei um Erbringer und Nutzer. Die Kommunikation zwischen beiden erfolgt durch Request/Response - Paare. Der Client erfragt beim Server die Erfüllung einer Aufgabe (Request). Der Server antwortet nach deren Erfüllung (Response).

Charakteristisch für Client-Server-Modelle ist die besondere Rolle bestimmter Rechner und ihre Zentralität. Der Server ist eine zentrale Instanz für die Clients. Er ist den Clients bekannt, und nur er erfüllt die gewünschte Aufgabe. Zentralität birgt, wie die meisten Eigenschaften, Vor- und Nachteile. Zentrale Instanzen sind oftmals bekannt bzw. ihr Auffinden gestaltet sich einfacher. Zudem wird die Wartung durch die Zentralität begünstigt. Schwieriger gestaltet sich allerdings die Erbringung der geforderten Verfügbarkeit. Nachteilig an

der zentralen Rolle des Servers ist ausserdem das Engpassrisiko, das mit der Diensterbringung für zahlreiche Clients einhergeht.

Auf Zentralität aufbauende Konzepte schränken demnach die Flexibilität ein.

1.2 Hierarchisch

Wenn ein Rechner zugleich Client und Server für andere Clients ist, spricht man von einer hierarchischen Struktur. Eine solche hierarchische Struktur ist zum Beispiel beim Domain Name System des Internets zu finden.

1.3 Peer-To-Peer

Das Client-Server-Modell, mit seiner Eigenschaft der Zentralität, ermöglicht die Nutzung der Vorteile von Rechnernetzen. Man kann sich nun fragen, ob durch flexiblere Rechnerverbunde weitere Potentiale erschlossen werden können.

Das Peer-To-Peer Modell (im weiteren P2P genannt) verfolgt den Ansatz eines flexiblen Rechnerverbundes. Rechner werden bei diesem Modell gleich behandelt. Es gibt demzufolge keine Rechner, die eine bestimmte herausragende Position, wie z.B. die eines Servers, im Verbund einnehmen.

Im Gegensatz zur Kommunikation mit einer zentralen Instanz, erfolgt die Kommunikation hier direkt unter den verbundenen Rechnern.

Die Rechner (Peers) nutzen die Ressourcen der anderen Peers und stellen eigene Ressourcen bereit. Dabei kann es sich z.B. um Datenbestände handeln. Interessante Anwendungen lassen sich etwa bei den File-Sharing-Tools im Internet beobachten.

Ein solches "Ressourcennetzwerk" könnte sehr dynamisch gestaltet werden. Die Ressourcenhaltung geschieht in P2P-Netzen auf den Endsystemen im Netzwerk statt im Zentrum. Für Endsysteme sind aber Anforderungen, wie ständige Verfügbarkeit, nicht erfüllbar. Durch die Dynamik der verbundenen Rechner ändern sich auch die im Netz enthaltenen Ressourcen. Der dynamische Charakter der P2P-Netze bringt viele Herausforderungen hinsichtlich der Gestaltung der Anwendungen mit sich.

Der dezentrale Charakter des P2P-Modells begünstigt unter anderem die Skalierbarkeit - dies ist ein wichtiges Kriterium bei Anwendungen mit ständig wachsendem Ressourcenbedarf - aber auch die Zuverlässigkeit (Redundanz).

Jedoch ergeben sich auch einige Schwierigkeiten durch das Fehlen einer zentralen Instanz. So wird der Einstieg von Rechnern ins Netz erschwert, da diese erst einen anderen Peer lokalisieren müssen, der bereits im P2P-Netz ist. Jeder Peer benötigt also zumindest einen anderen Peer um Informationen über

andere Rechner (und damit Ressourcen) im Netz zu erhalten.
Ebenfalls schwieriger gestaltet sich die Koordinierung von Teilaufgaben, die zur Erfüllung einer Gesamtaufgabe im Netz ausgeführt werden.

Es soll noch erwähnt werden, dass auch zentrale Server in P2P-Netzen zur Vereinfachung bestimmter Aufgaben eingesetzt werden können. Der Discovery-Prozess gestaltet sich einfacher, wenn eine zentrale Instanz ein Verzeichnis über die verbundenen Peers bereithält. Es handelt sich dann um Misch- oder Hybridvarianten des P2P-Konzeptes.

Neben dem bereits genannten File-Sharing sind Collaboration und verteiltes Rechnen weitere interessante Anwendungen, für die P2P-Netze eine Basis sein können.

In heutigen Netzen, z.B. im Unternehmen, Campus oder im Internet, sind viele Ressourcen ungenutzt auf den Endsystemen vorhanden. Das P2P-Modell ermöglicht einen Verbund dieser Endsysteme, um deren Ressourcen sinnvoll für komplexe Anwendungen zu nutzen.

2 Evolutionäre Algorithmen

Dieser Abschnitt soll als kurzer Überblick und Motivation zum Gegenstand der Evolutionären Algorithmen dienen.

Optimierungen sind essentiell für viele Bereiche in Wissenschaft, Technik und Wirtschaft. Zu zahlreichen praktischen Problemen ist eine optimale (im Sinne von bestmögliche) Lösung gefordert. Durch Nebenbedingungen wird der Prozess der Optimierung schnell sehr komplex. Die Bedeutung der Optimierung zeigt sich unter anderem durch die Menge an verschiedenen Verfahren.

Evolutionäre Algorithmen bezeichnen ein Klasse der stochastischen Optimierungsverfahren. Sie simulieren den Prozess der natürlichen Evolution auf einem Rechnersystem. Man kennt verschiedene evolutionsbasierte Verfahren. Im Wesentlichen sind dies Genetische Algorithmen, Evolutionäre Programmierung und Evolutionsstrategien. Die Verfahren arbeiten auf einer Menge von Lösungskandidaten, die sukzessive, in Anlehnung an die Evolutionsprinzipien Selektion und Variation, verändert werden.
Unter Selektion versteht man hierbei die Auswahl der jeweils besten Kandidaten zur Lösung des Problems aus der gesamten Menge. Die Auswahl erfolgt auf Basis von Reproduktionswahrscheinlichkeiten, die direkt mit der Qualität der Lösungen zusammenhängen. Die Qualität wird in Bezug auf die Zielfunktion bestimmt.
Das Prinzip der Variation umfasst die Rekombination und Mutation bestehender Kandidaten zu neuen Lösungen.

Die Simulation beider Prinzipien erlaubt die Entwicklung immer besserer Lösungen aus durchschnittlichen Anfangs- bzw. Zwischenlösungen.

Trotz der Einfachheit der zugrundeliegenden Konzepte, sind Evolutionäre Algorithmen robuste und mächtige Suchverfahren, die allgmein einsetzbar sind. Sie sind besonders für komplexe Suchräume und Probleme mit mehreren Zielfunktionen geeignet.

Die Entwicklung guter Lösungen mit evolutionären Verfahren erfordert eine hohe Anzahl verschiedener Individuen, auf denen die oben genannten Prinzipien simuliert werden können. Damit steigt gleichzeitig der mit der Berechnung verbundene Aufwand. Daher gab es schon frühzeitig Bestrebungen zur Parallelisierung Evolutionärer Algorithmen. Zur parallelen Ausführung evolutionärer Algorithmen sind verschiedene Herangehensweisen bekannt.
Beim Master-Slave Modell gibt es einen ausgezeichneten Knoten (Prozessor, Rechner), der die Kontrolle über den Algorithmus innehat. Die aufwendige Berechnung der Qualitätswerte einzelner Lösungen wird jedoch an die Slave-Knoten delegiert. Sämtliche Ergebnisse laufen daraufhin wieder beim Master zusammen.
Die andere Klasse unterscheidet in Coarse-Grained und Fine-Grained Parallelisierung. Die Parallelität beim Coarse-Grained-Ansatz wird durch die Ausführung mehrerer Evolutionärer Algorithmen auf jeweils eigenen Populationen (Demen) erreicht. Der Austausch von Lösungen zwischen diesen Demen begünstigt die Entwicklung guter Lösungen in den verschiedenen Populationen.

3 eaLib

Bei eaLib handelt es sich um eine Java-Klassenbibliothek zur Programmierung und Simulation evolutionärer Algorithmen. Die Bibliothek stellt Basisklassen für Daten und Operatoren bereit. Zusätzlich bietet sie die Funktionalität, komplexe Algorithmen aus einzelnen Komponenten zusammenzusetzen. Weiterhin werden Ausgabe- und Statistikfunktionen unterstützt.

4 Verteiltes Rechnen mit dem P2P-Modell

Das P2P-Modell ist bisher vor allem durch File-Sharing-Anwendungen im Internet bekannt geworden. Verschiedene Internet-Nutzer bilden hier, auf P2P-Basis, einen Verbund ihrer Rechner. Als Ressource wird anderen der Zugriff auf einige Dateien des eigenen Rechners gewährt. Die grosse Anzahl an Teilnehmern führt zu riesigen Datenbeständen in diesen Systemen, die mittels eigener Suchverfahren benutzt werden.

Ebenso interessant sind Anwendungen, welche eine P2P-Netz benutzen um Berechnungen in paralleler Form auszuführen. Einzelne Rechner, die Knoten in

einem P2P-Netz, können ihre Rechenleistung als Ressource anderen Rechnern zur Verfügung stellen. Eine grosse Anzahl von PCs, z.B. in einem Unternehmensnetzwerk oder Endsysteme im Internet, nutzen ihre Rechenleistung nicht vollständig und durchgehend. P2P-Netze stellen eine Möglichkeit dar, diese ungenutzte Leistung zahlreicher Systeme zu aggregieren und nutzbar zu machen. Projekte wie SETI@home im Internet verfolgen dieses Ziel. Hier werden die Rechner der Nutzer verwendet, um in Daten nach bestimmten Mustern zu suchen. Diese überaus rechenintensive Aufgabe wird durch die Aufteilung in kleinere Teile bewältigt.

Der P2P-Ansatz für verteilte Berechnung zielt auf die Nutzung der Vorteile des P2P-Modells ab. Die Dezentralität solcher Netze wird ausgenutzt, um die Toleranz gegenüber Ausfällen zu erhöhen und um Lasten besser zu verteilen.Die dynamische Eigenschaft von P2P-Netzen beeinflusst ebenfalls deren Eignung für derartige Berechnungen. So kann das Hinzukommen von Peers förderlich für die Ausführung der Berechnung sein, soweit diese beteiligt werden. Demgegenüber darf das Austreten von Peers aus dem Netzwerk keinen Einfluss auf die Ergebnisse haben. Gleichzeitig ist dieser dynamische Charakter eines der Merkmale, welches diese Systeme von bestehenden Systemen wie Multi-Prozessor-Rechnern unterscheidet.

Ein P2P-Netz als Basis für ein solches System begünstigt auch dessen Skalierbarkeit. Die Nutzung der Ressourcen, der Rechenleistung in diesem Fall, kann zum Beispiel in Abhängigkeit der Aufgabe geschehen.

Evolutionäre Algorithmen stellen einen Anwendungsfall dar, der geeignet ist die Möglichkeiten und die damit verbundenen Anforderungen eines solchen Ansatzes zu untersuchen. Ihre Eignung begründet sich, neben dem hohen Rechenaufwand für derartige Realanwendungen, aus der inhärenten Parallelität der Algorithmen. Es handelt sich dabei um eine generelle Anforderung an solche Anwendungen. Die Gesamtaufgabe muss in kleinere Teile zerlegbar sein. Das Gesamtergebnis muss sich wiederum aus den Teilergebnissen berechnen lassen. Bei den Teilaufgaben handelt es sich um Berechnungen, die auf einem lokalen Datenbestand arbeiten. Auf einen Peer übertragen bedeutet das, dem Peer wird eine Berechnungsvorschrift zugeteilt und Daten, auf denen er diese ausführt.
Der Evaluationsoperator der Evolutionären Algorithmen, der für die Bestimmung der Qualität von Lösungen zuständig ist, arbeitet auf lokalen Daten, ebenso die Operatoren für Variationen. Ihre Ausführung beschränkt sich jeweils auf einen Lösungskandidaten des Algorithmus. Selektion ist ein Beispiel für einen Operator, der global auf den Daten arbeitet. Um seine Entscheidung zu treffen, benötigt er eine Übersicht über die Reproduktionswahrscheinlichkeiten aller Lösungen.

4.1 Einige Konzepte

Um Berechnungen auf einem P2P-Netz auszuführen, muss eine entsprechende Infrastruktur bestehen. Sie umfasst die P2P- spezifische Funktionalität der Knoten des Netzes.

P2P-Netze, als dynamischer Verbund von Rechnern, setzen Möglichkeiten zum Aufbau und zur Verwaltung dieses Verbundes voraus. Da P2P-Netze auf direkten Kommuniaktionsverbindungen der Peers beruhen, stellen zwei Rechner, die einander kennen, bereits ein solches Netz dar. Die Aufnahme eines weiteren Rechners in das Netz kann nur erfolgen, wenn dieser Kenntnis von mindestens einem der beiden anderen Rechner hat. Diese neue direkte Verbindung zum bestehendem P2P-Netz ermöglicht dem Rechner das weitere Erforschen des Netzes. Unter dem Begriff Discovery werden Verfahren zusammengefasst, die darauf abzielen diese initiale Verbindung zum P2P-Netz herzustellen[1].

Verschiedenste Ansätze sind hierfür bekannt. Mit Broadcast-Verfahren informiert ein Peer seine nähere Umgebung über seine Existenz. Aus den empfangenen Antworten erhält er Kenntnis über Rechner im P2P-Verbund. Broadcast-Verfahren sind jedoch nur in lokalen Netzen einsetzbar. Ausserdem belasten sie das Netzwerk durch den erzeugten Verkehr.
Andere Mechanismen beruhen auf Voreinstellungen von gut bekannten Rechnern. Mit Hilfe dieser Listen wird versucht die Verbindung zu einem P2P-Netz herzustellen. Eine kontinuierliche Erweiterung und Aktualisierung der Listen sorgt für deren Tauglichkeit bei einer nächsten Discovery-Prozedur.

Ein zentrales Verzeichniss ist der einfachste Weg um Zugriff auf ein P2P-Netz zu erlangen. Es enthält Informationen über alle bereits verbundenen Peers und kann somit Auskunft über mögliche Einstiegspunkte geben. Nachteil dieser Variante ist die teilweise Zentralisierung des P2P-Modells. Einhergehend damit sind Aspekte wie Verfügbarkeit und Engpassrisiko des Verzeichnissdienstes.

Besteht diese initale Verbindung nach einen erfolgreichen Discovery-Vorgang, kann der Peer weitere Informationen über das Netz sammeln und sich über Veränderungen auf dem Laufendem halten.

Erweiterungen durch Hinzukommen neuer Peers und Verkleinerung durch Abgehen von Rechnern verändern das bestehende P2P-Netz ständig. Unter bestimmten Umständen kann es auch zur Partitionierung des Netzes kommen. Die Verbindungen im P2P-Netz werden in den Peers dadurch repräsentiert, dass sie ihr jeweiliges Gegenüber kennen. Fehlt diese Kenntnis, z.B. durch Ausstieg eines Peers, besteht auch diese Verbindung nicht mehr. Wenn es sich dabei um die einzige Verbindung zwischen zwei Teilbereichen des Netzes han-

[1] In der Literatur umfasst der Begriff Discovery manchmal auch die anschliessenden Schritte nach der Herstellung einer initialen Verbindung, also die Exploration des Netzwerkes.

delte, so werden diese getrennt. Es existieren jedoch auch Algorithmen, die die hinreichende Verbreitung der Verbindungsinformationen im Netz fördern, um derartige Fälle zu vermeiden.

Ein wichtiges Konzept, was auch von einigen Collaboration-Anwendungen genutzt wird, sind die Peer-Groups. Diese Gruppen sind Zusammenschlüsse von Peers innerhalb des Netzes nach bestimmten Gesichtspunkten. Es kann sich zum Beispiel um Peers handeln, die an der Erfüllung einer gemeinsamen Aufgabe arbeiten oder eine gleiche bzw. ähnliche Funktionalität aufweisen

Typischerweise wird eine verteilte Berechnung von einem Peer im Netz angestoßen. Es ist nun erforderlich die Aufgabe in separate Teile zu zerlegen und diese, jeweils mit der erforderlichen Berechnungsvorschrift, an andere Peers zu verteilen.
Für die evolutionären Algorithmen würde dies eine Zerlegung der Gesamtpopulation in Teilpopulationen bedeuten, die zusammen mit dem Algorithmus des jeweiligen Operators an einen Rechner übermittelt werden.

Es ist denkbar das der initiierende Peer in die Berechnung eingebunden ist. Er kann z.B. Koordinierungsfunktionen übernehmen. Eine weitere Variante sieht vor, das er nach der Initiierung der Aufgabe nur auf deren Beendigung wartet, ohne weitere Funktionenen zu übernehmen.

Peers können also jeweils den gleichen Funktionsumfang besitzen oder aber für bestimmte Aufgaben vorgesehen sein. Hierbei ist die Verwendung von Peer-Groups denkbar. So können Gruppen für die reine Berechnung, andere für Koordinierungsaufgaben gebildet werden. Das Kommunikationsaufkommen im gesamten Netz könnte durch Verwendung des Gruppenkonzeptes verringert werden, da Nachrichten nur innerhalb der betreffenden Gruppen bzw. über einzelne Verbindungen nach außen ausgetauscht werden. Peers, die einen Großteil ihrer Rechenleistung als Ressource im Netz anbieten wollen, würden sich in der Gruppe zusammenschliessen, die für die eigentliche Berechnung zuständig ist. Andere könnten sich an den Gruppen beteiligen, die weniger rechenaufwändig sind. Natürlich ist auch die Mitgliedschaft in mehreren Gruppen möglich.

Die Verwendung von Peer-Groups und die Bearbeitung bestimmter Aufgaben durch diese ist flexibler gegenüber der erstgenannten Variante. Werden die Peers innerhalb einer Gruppe durch geeignete Verfahren auf dem gleichen Stand gehalten, erzeugt diese eine Redundanz, mit der Ausfälle einzelner Rechner kompensiert werden können. Die Übernahme von Aufgaben durch mehrere Peers ist also eine wichtige Voraussetzung für die Entwicklung einer ausfalltoleranten Architektur. Desweiteren begünstigt eine allgemeine Definition der Gruppen und ihrer Aufgaben die Ausweitung des Ansatzes auf verschiedenste Berechnungsprobleme.

Discovery und Peer-Groups sind einige wichtige Konzepte von P2P-Netzen.

Ebenso wichtig sind Informationsprotokolle und Aspekte der Sicherheit in derartigen Netzen. Eine Infrastruktur, die diese Konzepte beinhaltet, ist Voraussetzung für die Entwicklung von verteilten Anwendungen auf Basis von P2P-Netzen.

Es soll noch darauf hingewiesen werden, dass verschiedene Projekte sich mit der Spezifikation und Implementation einer solchen, auf den letzten Seiten beschriebenen, Infrastruktur für P2P-Anwendungen befassen. Eines dieser Projekte ist das von SUN initiierte JXTA-Projekt. Bei JXTA werden den Entwicklern unter anderem die oben beschriebenen Dienste angeboten.

4.2 Berechnungen verteilen

Ziel eines System zur verteilten Berechnung ist es, eine komplexe Aufgabe durch Ausführung auf mehreren Teilsystemen zu beschleunigen. Diese Teilsysteme sind die Rechner eines P2P-Netzes, die sich dynamisch zusammenschliessen können. Die Funktionalität zur Verwaltung dieses Netzes wird durch eine P2P-Infrastruktur bereitgestellt. Aufbauend auf dieser sind Mechanismen nötig, welche der Verteilung von Daten und Befehlen (Code) dienen. Die Zusammenführung der Teilergebnisse der einzelnen Peers und die Gewinnung des Gesamtergebnisses stellt die Beendigung einer solchen verteilten Berechnung dar.

Die Verteilung von Berechnungen beinhaltet daher unterschiedliche Teilaufgaben:

- Definition der Aufgaben
  (job definition)
- Auswahl der Rechner zur Übermittlung
- Übermittlung der Aufgaben
  (job submission)
- eventuelle Statusabfragen
- Ergebnisgewinnung
  (result retrieval)

Die Erfüllung dieser Teilaufgaben erfordert die Kommunikation der Peers untereinander. Die P2P-Ebene sollte deshalb Möglichkeiten der Nachrichtenübermittlung bereitstellen.

Eine Aufgabe (job) besteht aus der Berechnungsvorschrift und den Daten. Eine typische Aufgabe im Zusammenhang mit evolutionären Algorithmen ist die

Berechnung der Qualität einer bestimmten Menge von Lösungen (Evaluation oder Fitneßberechnung). EaLib nutzt hierzu genetische Operatoren, welche durch Klassen definiert sind. Die Methoden der Klassen werden vom Entwickler problemspezifisch implementiert.
Die Verwendung von Klassen ist natürlich problemunabhängig. Klassen werden daher zur Beschreibung aller Aufgaben verwendet. Die Schnittstelle der Klasse spielt hierbei eine wichtige Rolle. Alle Aufgaben müssen für die Peers im Netzwerk gleich aussehen. Die Benutzung der Klasse muss ihnen möglich sein, egal welche Aufgabe die Klasse implementiert.

Steht die jeweilige Klasse auf den Peers zur Verfügung, werden die Methoden einer definierten Schnittstelle aufgerufen, um die Berechnung durchzuführen. Für eine solche definierte Schnittstelle bietet sich in Java-Umgebungen das Interface *Runnable* an. *Runnable* wird in den Java-Klassenbibliotheken definiert und steht somit auf jedem Peer mit der Java-Virtual-Machine zur Verfügung. Ein weiterer wichtiger Grund für die Unterstützung des *Runnable*-Interfaces durch die Klasse ist die Ausführung der Klasse in einem Thread auf dem Remote-Peer.

Eine Klasse, die das *Runnable*-Interface unterstützt, definiert die Method run(). Diese Methode wird von einem Thread auf dem Remote-Peer aufgerufen. Die Klasse kann durch Implementation dieser Methode ihre Berechnung starten.

Eine weitere Schnittstelle aus den Java-Klassen, das Interface *Serializable*, wird ebenfalls implementiert. Auf die Gründe wird weiter unten im Text eingegangen. Dieses Interface enthält keine Methoden, die vom Entwickler implementiert werden müssen. Es markiert die Klasse lediglich als serialisierbar.

Neben dem Starten einer Berechnung ist natürlich das Ergebnis von entscheidender Bedeutung. Die übertragene Klasse sollte daher Methoden bereitstellen, welche das Abfragen der Ergebnisse ermöglichen.

Das folgende Beispiel zeigt, wie die Definition einer Aufgabe aussehen könnte.

```
public class JobClass implements Runnable, Serializable
{
  private int result, a, b;

  public JobClass( int a, int b )
  {
    this.a = a;
    this.b = b;
  }

  public void run()
  {
    result = a + b;
  }

  public int getResult()
  {
    return result;
  }
}
```

Die Daten, auf denen eine Berechnung stattfinden soll, werden ebenfalls durch Klassen definiert. EaLib kennt dafür die Klasse **Individual**. Ein Individuum enthält ein ChromosomeSet, und dieses wiederum die Chromosomen. Individuen können verschiedene Arten von Chromosomen enthalten und fungieren daher als ein generischer Rahmen. Die genetischen Operatoren in eaLib arbeiten jeweils auf den Individuen (bzw. auf Feldern von diesen).

Vor einer Übertragung der Klassen müssen die Empfänger bestimmt werden. Die P2P-Schicht unterhält oftmals eine Liste der ihr bekannten Peers im Netz. Genauso kann der Verteilung eine Exploration des Netzes vorausgehen um mögliche Kandidaten zu entdecken. Die Anzahl der Teilaufgaben sollte sich nach der Anzahl der bekannten Peers richten. Sind mehr Peers als Aufgaben vorhanden, kann zum Beispiel eine Auswahl stattfinden. Dafür kommen Kriterien, wie Entfernung und Zuverlässigkeit in der Vergangenheit, in Betracht. Dazu ist es nötig, solche Informationen laufend zu erzeugen. Die P2P-Schicht könnte dies etwa durch entsprechende Protokollierung tun.

Die Organisation des P2P-Netzes kann auch gesonderte Peers für die Job-Verteilung vorsehen (Peer-Gruppe Dispatcher). In diesem Fall ist der intiierende Peer selbst nicht mit der Verteilung befasst. Er übermittelt lediglich die Teilaufgaben an diese Gruppe. Ein Rechner muss sich somit nicht selber mit der Auswahl der Peers befassen, sondern überlässt dies den Job-Dispatchern.

Klassenübertragung

Die vorherige Übertragung der Klassen auf einen anderen Rechner unterscheidet die verteilte Ausführung von der lokalen. Ein Konzept, welches direkt von der Java-Virtual-Machine unterstützt wird, ist das Classloader-Konzept. Die JVM benutzt es um referenzierte Klassen zu laden. Die lokale Variante lädt die Klasse von der Festplatte in die JVM.

Auch das Netzwerk kann als Quelle für Klassen dienen. Entsprechende **Classloader** werden durch die Java-Bibliothek bereitgestellt. Es ist aber auch möglich den Klassenlader selber zu implementieren. Dadurch wird es dem Entwickler ermöglicht, den Ladevorgang an die Bedürfnisse anzupassen.

Mittels Klassenladern kann die Übertragung und Ausführung von Klassen aus dem Netzwerk bewerkstelligt werden. Die geladene Klasse enthält den Code der Berechnungsvorschrift. Die zur Aufgabe gehörenden Daten können über Objekt-Serialisierung übertragen werden. Die Art der Daten ist von Aufgabe zu Aufgabe verschieden. So sind es bei evolutionären Algorithmen z.B. Objekte der Klasse **Individual**. Das bedeutet aber, daß ein Remote-Peer, der die Objekte deserialisiert, die Klassendefinition der Daten benötigt. Auf jedem Remote-Peer wird also zum Beispiel die Klasse **Individual** benötigt. Für eine auf eaLib spezialisierte Architektur würde dies ausreichen, aber eine generelle Verwendung des Ansatzes wird dadurch erschwert. Wie oben erwähnt sollen Remote-Peers eine Aufgabe ausführen können, ohne allzuviel darüber zu wissen. Inwieweit sich dies realisieren lässt, soll im Folgenden kurz erläutert werden.

Um die Bearbeitung einer Aufgabe auf einem Remote-Peer durchzuführen, muss dieser die Konzepte Klassenlader und Serialisierung verbinden. Die Klassen definieren den Code, und die Objekte enthalten die jeweiligen Daten. Eine dynamische geladene Job-Klasse lässt sich instantiieren und ausführen. Um sie aber sinnvoll zu nutzen, muss sie mit den Daten des Serialisierungsstroms initialisiert werden. Eine Möglichkeit, die dies eventuell bewerkstelligt, ist die Benutzung von Klassen-Annotationen. Diese Erweiterung des Serialisierungskonzeptes sieht vor, dem serialisierten Objekt Informationen über seine Klasse mitzugeben. Diese Informationen können z.B. der Byte-Code der Klasse oder eine URL sein. Auf der Deserialisierungsseite kann eine Resolver-Implementation die Annotationen nutzen, um das Objekt zu erstellen. Diese Methode erfordert das Überschreiben einiger Standard-Methoden der Klassen **ObjectStreamOutput** und **ObjectStreamInput** des Java-API.

Die Nutzung von Annotationen konnte bis zum jetzigen Zeitpunkt noch nicht verwirklicht werden. Deshalb kann an dieser Stelle noch keine Einschätzung über deren Tauglichkeit gegeben werden.[2]

[2]Die Nutzung von Annotationen bei der Serialisierung wird zum Beipiel durch Java-RMI implementiert.

Die Deserialisierung beliebiger Objekte lässt sich dennoch realisieren. Die hier benutzte Technik ist unter Bootstrapping bekannt. Will ein Peer aus einem Serialisierungsstrom ein Objekt eines bestimmten Typs erstellen, so benötigt er die Klasse dieses Typs. Die JVM benutzt zum Laden dieser Klasse den Kassenlader, der die Klasse, welche jetzt den Typ benötigt, geladen hat. Oder anders: Der Klassenlader, der die Klasse des Client-Programms auf dem Peer geladen hat, lädt auch alle in ihr referenzierten Klassen. Dies ist oftmals der Klassenlader für das lokale Dateisystem. Mittels Bootstrapping kann ein Klassenlader bestimmt werden, der sich alle benötigten Klassen von einer vorgegebenen Stelle besorgt, zum Beispiel im Netzwerk.

Beim Bootstrapping wird das Client-Programm selber aus dem Netzwerk geladen. Jeder in ihm referenzierte Typ wird von der gleichen Stelle im Netzwerk beschafft, wenn er nicht lokal vorhanden oder schon geladen ist. Auf dem Peer brauchen damit aufgabenspezifische Klassen nicht vorhanden zu sein. Dies ist eine der Anforderungen, die wir an ein solches System gestellt haben. Die aus dem Netz geladene Client-Klasse ist aufgabenabhängig, und wird durch den Job-Initiator bereitgestellt. Man kann sie als eine Art Thread auffassen, der im Netz definiert und lokal ausgeführt wird.

Die Beispielklasse weiter oben stellt die Methode getResult() bereit. Mit dieser Methode kann das Ergebnis der Berechnung abgefragt werden. Zuvor muss die Klasse zurück zum Initiator gesendet werden. Die Deserialisierung gestaltet sich hier einfach, da alle Klassen der benötigten Typen lokal vorhanden sind.

Auf den letzten Seiten wurde eine Überblick über die Probleme und Ansätze zur Verteilung von Berechnungen gegeben. Bei der Implementierung sind eine Reihe anderer Fragen zu beantworten, auf die hier nicht eingegangen wurde.

4.3 Alternative Ansätze

Die Verteilung von Berechnungsvorschriften und Daten wurde bisher mittels eines Klassenladers und dem Konzept der Objektserialisierung erreicht. Es wurden auch Überlegungen hinsichtlich anderer Ansätze angestellt. Sie sollen in diesem Abschnitt kurz erwähnt und Gründe für deren Verwerfung genannt werden.

Statt eine Berechnung in einer Klasse zu definieren und diese an andere Rechner zu übertragen, wäre es auch denkbar diese Klasse erst auf dem Remote-Peer zu erzeugen und auszuführen. Diese Variante setzt allerdings voraus das dem Remote-Peer eine Beschreibung der Berechnung geliefert wird aus der er den konkreten Code erzeugt. Der Remote-Peer stellt gewissermassen eine Fabrik für Klassen bereit, die auf einer algorithmischen Beschreibung der Berechnung arbeitet.
Eine passende Beschreibungsprache müsste aber sehr umfangreich ausfallen um alle möglichen Berechnungen und Daten abzudecken, oder sie wäre nur auf ein spezielles Problem zugeschnitten. Der Entwurf einer solchen Beschreibungsmöglichkeit und die Erzeugung der Klasse auf dem Remote-Peer würden den Aufwand dieser Variante sehr erhöhen. Das Klassenladerkonzept wurde deshalb vorgezogen.

Für die Übertragung der Daten auf denen eine Berechnung stattfinden soll lassen sich sicherlich effizientere Methoden als Objektserialisierung finden. Sind die Daten auf beiden Peers wohlbekannt, könnten z.B. spezielle Formate ohne Overhead verwendet werden. Jedoch ist auch dies in zweierlei Hinsicht nachteilig. Die Flexibilität bei den Daten ginge verloren, da jetzt auf beiden Seiten Informationen über deren Aufbau vorliegen muss. Ausserdem werden Daten und Berechnungsvorschrift nun wieder getrennt betrachtet. So muss hier ein Weg gefunden werden die Berechnung auf den empfangenen Daten auszuführen. Das führt aber unter anderem zu den Problemen die im vorherigen Abschnitt erläutert wurden.

Der vorgestellte Ansatz umgeht diese Nachteile dadurch, dass Daten und Berechnung in einem Objekt zusammengefasst und an einen Remote-Peer übertragen werden. Der Remote-Peer benötigt zur Ausführung keinerlei Informationen über den Aufbau der Klasse sondern benutzt eine definierte Schnittstelle. Diese Black-Box-Herangehensweise stellt die Flexibilität der Berechnungen sicher.

4.4 Integration mit eaLib

In diesem Abschnitt soll es um die Verwendung von eaLib in einer Umgebung wie der oben beschriebenen gehen. Wie könnten die Vorteile der verteilten Berechnung für diese Bibliothek genutzt werden?
Ein Szenario in diesem Zusammenhang sieht z.B. folgendermaßen aus:
Auf einem Rechner wird eaLib zur Entwicklung eines Evolutionären Algorithmus verwendet. Da das Problem sehr umfangreich ist, kommen viele Lösungskandidaten (Individuen) in der Berechnung zum Einsatz. Zur Beschleunigung der aufwendigen Fitneßwertberechnung der einzelnen Lösungen soll diese, mittels eines P2P-Netzes, verteilt berechnet werden. Dazu wird die Gesamtmenge an Individuen in mehrere Teile zerlegt und zusammen mit dem Operator zur Fitneßermittlung an andere Peers verteilt. Nach Beendigung der Berechnung wird der Algorithmus lokal fortgesetzt.

Algorithmen bestehen in eaLib aus einer Folge von genetischen Operatoren. Die Verbindungen dieser Operatoren erfolgt durch den Entwickler und kennzeichnet die Ausführungsreihenfolge. Die Basisklasse aller genetischen Operatoren ist **GeneticOperator**. Aus dieser leiten sich Operatoren für Mutation, Selektion, Evaluation usw. ab. Die genannten Operatoren implementieren die Schnittstelle *CollectionProcessor*, welche die Methode process() enthält. Das Argument und der Rückgabewert von process() ist ein Feld von Individuen. Die process()-Methode startet die Ausführung des genetischen Operators auf den einzelnen Individuen[3].

Für die Repräsentation des Algorithmus baut eaLib intern einen Graph auf. Die Knoten dieses Graphes werden durch **FlowVertex**-Elemente beschrieben. Jeder Knoten enthält ein **FlowElement**. Ein **FlowElement** repräsentiert die unterschiedlichen Arten von Elementen, unter anderem ein **ConnectorElement**. Der Zusammenhang zwischen den genetischen Operatoren, welche der Entwickler erstellt, und den Elementen des Graphen, wird über eine Fabrik namens **FlowElementFactory** hergestellt. Sie erzeugt für jeden Operator ein passendes Element im Graphen. So enthält z.B. ein **ConnectorElement** eine Referenz auf ein *CollectionProcessor*-Interface.
Ein solcher Graph ermöglicht u.a. eine Überprüfung des Algorithmus auf eventuelle strukturelle Fehler.

Die Ausführung besteht nun in einem Durchlaufen des Graphen, wobei, beginnend bei den Knoten des Graphen (**FlowVertex**), eine Reihe von Methodenaufrufen stattfinden, die letztlich im Aufruf von process() enden.

[3] process() selber ist nur der Einstiegspunkt. Aber sie ruft Methoden auf, die durch eaLib selber oder den Entwickler des Algorithmus vorgegeben werden.

Um die Verteilung zu ermöglichen, muss die Klasse (**ConnectorElement**), welche die process()-Methode des jeweiligen Operators aufruft, entsprechend angepasst werden. Statt eines direkten Aufrufes müssen die weiter oben aufgezählten Teilschritte durchgeführt werden. Dies setzt natürlich das Vorhandensein der entsprechenden Infrastruktur (P2P-Schicht) voraus.
Diese Anpassung von eaLib hat zur Folge, dass alle Operatoren des Typs *CollectionProcessor* verteilt werden. In manchen Fällen ist dies aber vielleicht unerwünscht oder überflüssig. Im Falle der Selektion können auch Schwierigkeiten auftreten. Flexibler ist die Anpassung der jeweiligen Operatoren an eine verteilte Ausführung. Eine verteilungsfähige Variante des Fitneßoperators könnte in seiner process()-Methode entsprechende Aufrufe vornehmen.
Da die genetischen Operatoren das Interface *Runnable* nicht implementieren, muss dies nachträglich geschehen[4].

Das Ausührungsmodell von eaLib lässt diese Anpassungen an verschiedenen Stellen zu. Einige Anpassungen zeichnen sich durch Einfachheit und Generalität aber auch durch Einbüßen von Flexibilität aus. Andere benötigen eine umfassendere Anpassung der EA-Bibliothek an die Verteilung.

[4]Das bestehende Interface ist genauso einfach wie *Runnable*, würde aber die Verwendung der Architektur auf eaLib beschränken

5 Schlusswort

Im Text wurden grundlegende Aspekte zu P2P-basierten Berechnungsumgebungen vorgestellt. Die dargestellten Erkenntnisse sind keinesfalls vollständig, sondern dienen als Grundlage für weiterführende Entwicklungen. Es soll hier noch einmal erwähnt werden, dass es sich bei dem dargestellten Ansatz nicht um die einzige Möglichkeit handelt verteilte Berechnungen zu realisieren.

Ein wichtiger Punkt im Zusammenhang mit Netzwerken und verteilter Berechnung ist das Sicherheitskonzept. Obwohl im Text darauf nicht eingegangen wurde, bildet es einen wesentlichen Bestandteil einer Implementation. Es muss gewährleisten, dass Teilnehmer im P2P-Verbund vor unberechtigten Zugriffen und Manipulationen geschützt werden.
In diesem Zusammenhang ist noch ein anderes Problem offensichtlich. Da Teilergebnisse von verschiedenen, oftmals unbekannten Rechnern erbracht werden, kann deren Korrektheit nicht garantiert werden. Für einige Anwendungen kann dies problematisch sein. Hier könnten Mehrfachberechnungen und Vergleiche zumindest in Teilen die Sicherheit erhöhen.

Obwohl die einzelnen Teilsysteme ihre Berechnung weitestgehend selbständig ausführen erfordert die Gesamtaufgabe eine Kommunikation unter den Rechnern zum Zwecke der Koordination und des Datenaustausches. Dafür ist eine geeignete Kommunikationsarchitektur nötig.

Trotz all dieser Herausforderungen bei der Realisierung, können mittels des P2P-Ansatzes für verteilte Berechnungen bestehende Ressourcen besser genutzt werden. In vielen Teilen kann auch auf vorhandenes und erprobtes Wissen aus anderen Bereichen zurückgegriffen werden. Jedoch besteht auch Raum und Notwendigkeit für die Erforschung neuartiger Ansätze.

6 Anhang

6.1 Code

Auf den folgenden Seiten ist der Quelltext (Java) eines Demonstrators abgebildet. Er implementiert den besprochenen Ansatz zur verteilten Berechnung einer Aufgabe. Der Quelltext konzentriert sich auf die eigentliche Funktionalität im Zusammenhang mit der übertragung und Ausführung der Klassen. Als Aufgabe wurde dir Addition zweier Zahlen gewählt. Die zu summierenden Zahlen werden einem Programm übergeben, welches daraufhin die aufgabenbeschreibende Klasse erzeugt. Mit der Ausführung eines weiteren Programmes (auf einem anderen Rechner) wird diese Klasse geladen und ausgeführt. Das Ergebnis wird zurück an den initiierenden Rechner übertragen und dort ausgegeben. Die Kommunikation erfolgt das Socket-API von Java.

```
public class AddServer {
  public static void main(String args[]) {

    int port = 2233;
    InputStream input;
    OutputStream output;
    Socket socket;
    JobClass jc;

    // Argumente "ubernehmen
    int a = Integer.parseInt(args[0]);
    int b = Integer.parseInt(args[1]);
    System.out.println("first arg: " + a.getVal());
    System.out.println("second arg: " + b.getVal()+"\n");

    // Server starten und auf Verbindungen warten
    try {
      ServerSocket ssock = new ServerSocket(port);

      while (true) {
        socket = ssock.accept();           // Verbindung annehmen

        output = socket.getOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(output);

          // Job-Klasse serialisieren und "ubertragen
          jc = new JobClass(a,b);
          oos.writeObject(jc);
```

```
          oos.flush();
         oos.close();
        socket.close();
        break;
      }
      // Ergebnis empfangen
      while (true) {
        socket = ssock.accept();
        input = socket.getInputStream();
        ObjectInputStream ois = new ObjectInputStream(input);

           jc = (JobClass)ois.readObject();      // Object deserialisieren

        ois.close();
        // Ergebnis ausgeben
        System.out.println("Result: "+sc.getResult().getVal());
        socket.close();
        break;
      }
    }
    catch (IOException e) {
      System.out.println("!IO Exception!");
    }
    catch (ClassNotFoundException e) {
      System.out.println("!Class Not Found!");
    }
  }
}

public class AddTester {
  public static void main(String[] args) throws Exception {

    URL urls[] = { new URL("ftp://user:pass@somehost/dir_of_classes/") };
    int port = 2233;

    ClassLoader cl = new URLClassLoader(urls);
    Class c = cl.loadClass("AddThread");

      Runnable r = (Runnable)c.newInstance();
      r.run();
  }
}
```

```
public class AddThread implements Runnable {
  public void run() {

    int port = 2233;
    String host="somehost";
    InputStream input;
    OutputStream output;
    Socket sock;
    JobClass jc;

    try {
      sock = new Socket(host, port);       // verbindung aufbauen
      input = sock.getInputStream();
      ObjectInputStream ois = new ObjectInputStream(input);

      try {
        jc = (JobClass)ois.readObject();  //Job-Klasse empfangen
        jc.run();                         //Job ausf"uhren

        ois.close();
        input.close();
        sock.close();

        sock = new Socket(host,port);     // Ergebnis senden
        output = sock.getOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(output);
          oos.writeObject(jc);

        oos.close();
        output.close();
        sock.close();
      }
      catch (ClassNotFoundException e) {
    System.out.println("!Class Not Found!");
      }
    }
    catch (IOException e) {
      System.out.println("!IO Exception!");
    }
  }
}
```

6.2 Referenzen

http://www.java.sun.com Webseite für JAVA

http://www.jxta.org Webseite des JXTA-Projektes

Evolutionäre Algorithmen, Karsten Weicker, Teubner 2002

"A Survey of Parallel Genetic Algorithms",
Erick Cantu-Paz, University of Illinois

Peer-to-Peer Computing,
Dejan S. Milojicic, HP Laboratories Palo Alto

Efficient Topology-Aware Overlay Network,
Marcel Waldvogel, Roberto Rinaldi, IBM Research