Nutzen von p2p Techniken für die E/A in verteilten Rechnerumgebungen

Autor: Arne Babenhauserheide, Betreuung: Julian Kunkel

10. September 2008

Zusammenfassung

In this report we analyzed potential usage of p2p networks for efficient data replication in distributed systems.

Theoretical analysis of an optimal p2p replication structure shows a huge potential for improvements in the speed of data replication using a distributed system where each node in the network uploads and downloads with full speed at the same time.

We gathered and evaluated data in 7 nodes of our test cluster interconnected by a 1Gbps network and for a simulated WLAN environment.

However measured performance sticks behind theoretical considerations.

1 Gliederung

Inhaltsverzeichnis

2 Einstieg

Einstieg
In meinem Praktikum habe ich mich mit der möglichen Anwendung von p2p Techniken in verteilten Rechnersystemen beschäftigt.
Spezifischer habe ich die folgende Fragestellung behandelt:

2.1 Fragestellung

Fragestellung
"In welcher Umgebung bringen aktuelle p2p Techniken Vorteile bei der Dateiverbreitung in verteilten Rechnerumgebungen?"

2.2 Hintergrund

Bevor ich zur genauen Beschreibung gehe, werde ich ein paar Hintergründe des Praktikums beschreiben.

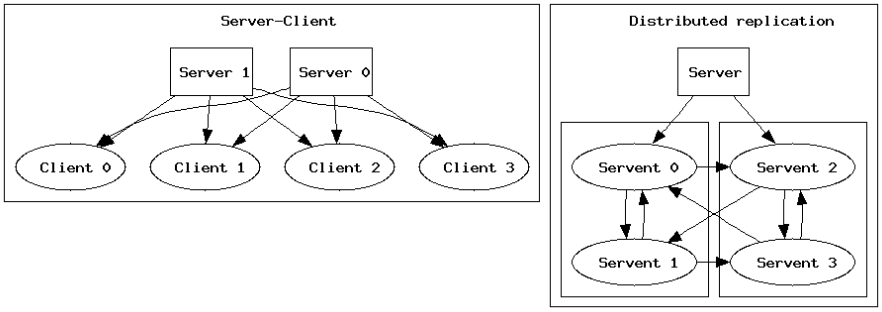
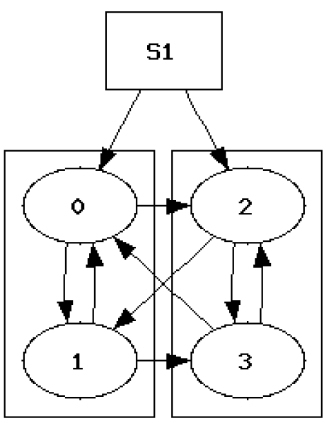


Abbildung 1: Gegenüberstellung: Ein verteiltes Dateisystem mit mehreren Datenservern und ein System mit verteilter Dateivervielfältigung

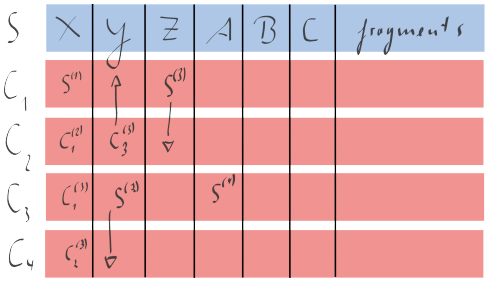
2.2.1 Skizze einer möglichen Funktionsweise eines idealen Netzes

Wie in 2 und 3 gezeigt überträgt in einem möglichen idealen Schema für die dezentrale Dateivervielfältigung ein Server die Daten nur an einige Knoten, die ihre Daten sofort an weitere Knoten verbreiten.
In dem in den Skizzen in 3 gezeigten, sehr vereinfachten Beispiel mit 4 Zielknoten überträgt der Server immer abwechselnd Fragmente an die Knoten C_1 und C_3, die dann ihr Fragment erst an denjenigen Nachbarknoten weitergeben, der gerade keine anderen Daten empfängt und es im nächsten Schritt an einen der beiden anderen Knoten übertragen (während ihr Nachbarknoten die Daten an den verbleibenden Knoten weitergibt).

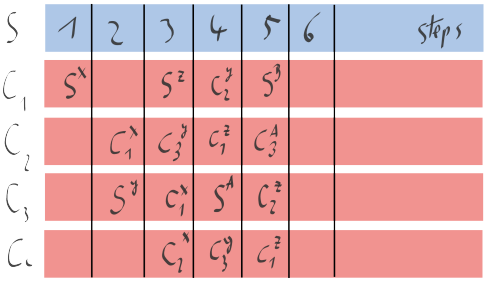


Example structure of ideal decentral file replication.

Abbildung 2: Dezentrale Dateivervielfältigung



Process of the transfer by fragment: "Who gets which fragment from whom?"



Process of the transfer by time: "When does which fragment get transfered by whom?"

Scheme of a way of chunked data transfer in a theoretical ideal p2p network.

Abbildung 3: Skizze der Verteilung der Daten in einem theoretischen, idealen p2p System

Die Skizze stellt keinen formellen Beweis dar, sondern ist einfach ein gedanklicher Entwurf für ein mögliches System, in dem alle Knoten immer ihre maximale Uploadleistung nutzen.
Die Anlaufzeit steigt hier mit O(log(N)); N: Anzahl der Knoten.

2.2.2 Einfachere Alternativen

Grundlegend gibt es die Möglichkeit, Daten in Einzelübertragungen zu den Zielrechnern zu bringen, oder jeden einzelnen Rechner zu einem Dateiserver für alle anderen zu machen.
Dazwischen gibt es allerdings weitere, einfachere Strukturen, die in kleineren Systemen genutzt werden können.

2.2.3 Übertragung in Reihe

Ein Beispiel, das in unserem Cluster wegen der geringen Anzahl genutzter Knoten (7 Zielknoten, auf die die Daten übertragen wurden) wohl die beste Leistung gebracht hätte, ist die Übertragung in einer Reihe, bei der alle Knoten als hintereinander geschaltet gedacht werden und der Dateiserver die Datei in kleinen Fragmenten an den ersten Downloader weitergibt. Dieser erste Downloader gibt dann jedes Fragment direkt an den nächsten in der Reihe weiter und speichert es gleichzeitig. Der nächste gibt es wieder weiter, bis der letzte in der Reihe erreicht ist (siehe 4).

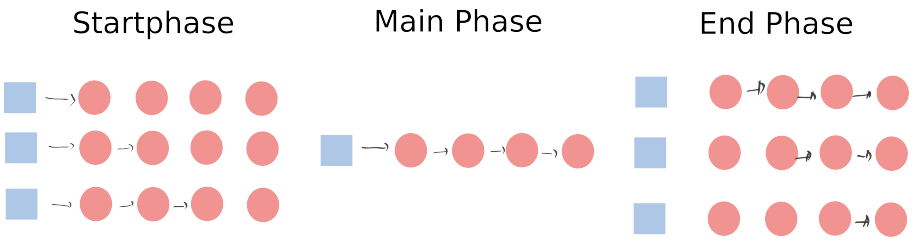


Abbildung 4: Verteilung von Daten in Reihe

Auf diese Art wird nach einer kurzen Anlaufzeit (Fragmentgröße / Einzelübertragungsrate Anzahl der Downloader) die Einzelleistung aller Beteiligten Knoten (bis auf die des Letzten) vollständig genutzt, und die einzelnen Fragmente können direkt im Speicher gehalten werden, so dass kein (langsamer) Festplattenzugriff für die Weitergabe notwendig sein sollte. Die erreichbare Gesamtgeschwindigkeit ist damit die Anzahl aller Downloader multipliziert mit dem Kleineren aus der Einzelleistung des Uploaders (Festplatte) und der Einzelleistung der Knoten (Netzwerk):

$$Geschwindigkeit = N_{DL} * min(Geschw.UL, Geschw.DL)$$

Die (langsamere) Startphase entspricht hier gerade der Anlaufzeit des Systems, also

$$Fragmentgroesse/Einzeluebertragungsrate * AnzahlderDownloader$$

und steigt so linear mit der Anzahl von Downloadern, während sie in einem idealen p2p System nur mit O(log(N)) steigt. Das gleiche gilt für die Endphase. Allerdings würde hier ein Bruch der Kette die gesamte Datenübertragung stoppen, und auch Last auf einzelnen Rechnern könnte alle in der Kette weiter hinten kommenden verzögern.

2.3 Vergleich: Was leisten verteilte Dateisysteme im Read-Only Modus?

Um diese Leistung in Relation zu bisher genutzten Systemen zu bringen, können wir sie mit der Leistung von verteilten Dateisystemen vergleichen.
Hier liefert meist ein Verbund von Rechnern die Daten an Zielrechner, so dass bereits ein sehr hoher Durchsatz gewährleistet wird.
Die maximal mögliche Leistung ist dabei

$$s_{ges} = AnzahlServer * Einzelleistung$$

Solange das Netzwerk also höchstens diese Leistung liefern kann, sind verteilte Dateisysteme und ideale p2p Netze gleichauf. Sobald aber die Gesamtleistung des Netzes deutlich darüber liegt (wie es z.B. bei sehr großen Clustern der Fall sein kann, genauso aber auch in WLAN Meshes) können p2p Systeme eine deutlich bessere Leistung bei der Dateiverteilung liefern, nämlich gerade Anzahl Knoten * Einzelleistung gegenüber Anzahl Server * Einzelleistung bei verteilten Dateisystemen. Vor allem aber benötigen ideale p2p Systeme nur eine einzelne erste Quelle und keine dedizierten Dateiserver.
Dabei ist allerdings zu bedenken, dass verteilte Dateisysteme noch viele andere Aufgaben übernehmen als die Vervielfältigung von Daten.
Zurück zu den p2p Techniken.

2.3.1 Herkunft der Techniken

Als nächstes interessiert mich hier, in welcher Umgebung sich diese Techniken entwickelt haben.
Was heute p2p Netzwerke genannt wird, gibt es nun effektiv seit 2001, als die Musikindustrie Napster mit Klagen überzog und aus den Tiefen von AOL und Nullsoft Gnutella die Bildfläche betrat, nach nichtmal einem Tag gelöscht und binnen der nächsten Monate von verschiedenen Projekten Reverse Engineered wurde.
Seitdem sind die p2p Netzwerke immer weiter verbessert worden, zum Gutteil, um sinnvolle Skalierbarkeit zu ermöglichen.

Im Verlauf dieser Anpassungen wurden hochspezialisierte Lösungen für die Dateiverteilung entwickelt, die effizientes Suchen und effiziente Dateiübertragung in Netzwerken mit über 5 Millionen gleichzeitigen Nutzern ermöglichen.
Und diese Techniken haben inzwischen gut 7 Jahre Entwicklungszeit und Tests durch Millionen von Nutzern in praktischem Einsatz hinter sich.
Die Umgebung, in der sich diese Techniken entwickelt haben, lässt sich dabei weitreichend durch zwei Parameter beschreiben:

- Es gibt eine starke Fluktuation der Beteiligten und
- Der Flaschenhals ist die Einzelleistung jedes Beteiligten, genauer: Die Uploadgeschwindigkeit, also die Einzelleistung der Netzwerkanbindung der beteiligten Rechner.

Starke Fluktuationen gibt es es im Hochleistungsrechnen bisher nicht, die Einzelleistung als Flaschenhals wird aber Schritt für Schritt zur Realität, wenn auch nicht durch die Netzanbindung, sondern durch die E/A.

2.3.2 Situation im Hochleistungsrechnen

Die Einzelleistung der E/A wird in einer Hochleistungsumgebung durch die E/A Leistung jedes einzelnen Systems begrenzt.
E/A ist hier die Datenleistung, die aus einem Verbund heraus geht, also das Minimum aus Netzwerkleistung des Verbundes und Festplattenleistung.
Und gerade die Plattenleistung fällt gegenüber der Gesamtleistung des Netzwerkes immer mehr zurück, so dass die Einzelleistung zum Flaschenhals wird.
Dadurch wandelt sich die Umgebung und der Einsatz von Techniken aus bestehenden p2p Netzwerken wird sinnvoller.
Zu den technischen Vorraussetzungen (Netzwerk schneller als Platten) kommt noch, dass die E/A meist von einem einzelnen Verbund geleistet wird, so dass die Einzelleistung dieses Verbundes die E/A Geschwindigkeit des gesamten Grids/Clusters begrenzt.
In dieser Situation habe ich mich in meinem Praktikum um drei grundlegende Fragen gekümmert.

2.3.3 Behandelte Fragen

Als Erstes habe ich die verschiedenen p2p Systeme betrachtet, die aktiv genutzt werden und damit erprobt sind. Für das Hochleistungsrechnen sind dabei meiner Einschätzung nach die folgenden Fragen am wichtigsten:

- Welche Anforderungen haben die verschiedenen p2p Systeme an “Dateiserver” und “Clients”?
- Wie gut funktionieren die p2p Systeme in LAN-Umgebungen? Wie stehen sie im Vergleich zu parallelen Dateisystemen für READ-only Daten da?

Die dritte und mir wichtigste Frage, die ich zu beantworten versucht habe, ist:

- "In welchen Umgebungen lohnt sich der Einsatz von p2p Systemen für die Dateiverbreitung?"

Als Antwort auf diese Frage habe ich versucht, konkrete Schwellen zu finden, an denen leicht geprüft werden kann, welches System sich in einer bestimmten Umgebung lohnt.
Im Verlauf des Praktikums habe ich, um stabilere Daten zu erhalten, diese Fragestellung auf die einfachere Frage begrenzt:
"Ab welcher Datenmenge ist der Einsatz von p2p Techniken sinnvoll?"

3 Hintergrund zu den Techniken

3.0.1 Wie funktionieren aktuelle p2p Systeme?

Die Systeme, die im aktuellen Sprachgebrauch als p2p Systeme bezeichnet werden, dienen v.a. der Dateiverteilung.
Dabei wird eine Datei nicht mehr nur von einem Server an einen Client übertragen. Stattdessen tauschen die Clients (zusätzlich) untereinander Dateifragmente aus, so dass es oft genügt, wenn von den zu übertragenden Daten nur etwas mehr als eine vollständige Kopie direkt von der ersten Quelle hochgeladen wird.
Clients werden hier oft als Knoten bezeichnet, da sie Aufgaben von Servern mit übernehmen. Der Server wird so teils zur ersten Quelle der Datei, teils sogar zum reinen Koordinator.
Übertragungen werden damit bei zunehmender Anzahl von Knoten immer unabhängiger von der Bandbreite der ersten Quelle.
Zusätzlich lassen sich aktuelle p2p Systeme grob in zwei Kategorien einteilen:

- Echt dezentrale Systeme
- Teilzentralisierte Systeme

Echt dezentrale Systeme sind beispielsweise Gnutella und verschiedene Kademlia Implementierungen, bei denen sich die Knoten im Netzwerk untereinander verbinden und ohne eine zentrale Kontrollstelle Dateien und Informationen über Dateien autauschen.
Dabei hat jeder Knoten ein begrenztes Wissen über das Netzwerk, das er an andere weitergibt.
Auf diese Art gibt es in echt dezentralen Systemen keinen Single Point of Failure und keine Adresse, zu der jeder Knoten eine Verbindungsmöglichkeit haben muss.
Im Gegensatz dazu nutzen teilzentralisierte Systeme eine zentrale Stelle, die mehr oder weniger Aufgaben für das Netzwerk übernimmt.
In dem hier untersuchten System BitTorrent übernimmt ein BitTorrent Server die Koordination der Verteilung einer Datei, allerdings kann für jede einzelne Datei ein anderer Server genutzt werden.
Der Server kennt hier jeden Knoten und die Knoten erhalten ihre Informationen über andere Knoten (fast) nur von dem zentralen Server, der ausserdem

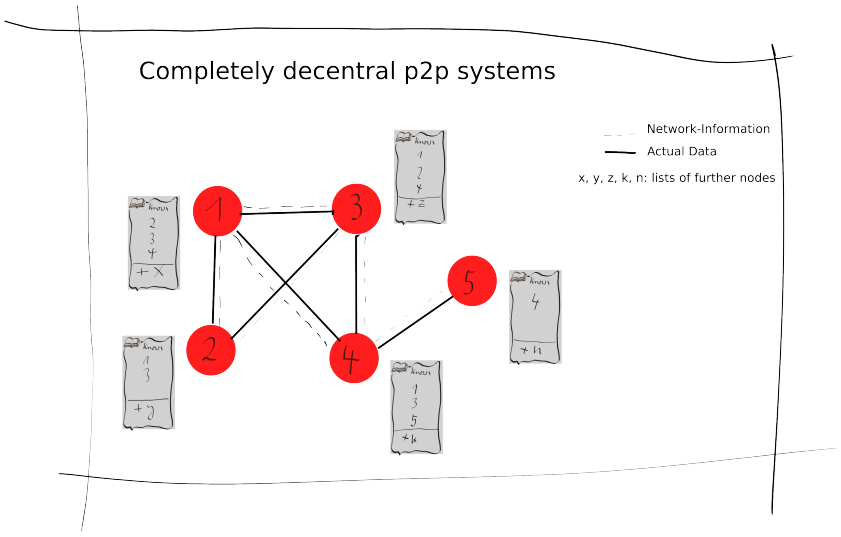


Abbildung 5: Struktur von vollständig dezentralen p2p Netzwerken

Informationen liefert, wieviele Daten die einzelnen Knoten hochladen, so dass andere Knoten ihr Verhalten daran anpassen können.
Aus diesem Grund braucht allerdings jeder teilnehmende Knoten eine stabile Verbindung zum Server.
Im Rest des Praktikums bezeichne ich jeden an der Übertragung Beteiligten Rechner einfach als Knoten im Netz oder als „Uploader“, wenn er nur Daten liefert, als „Downloader“, wenn er sowohl Daten empfängt als auch sie verteilt oder als Server, wenn auf ihm nur ein BitTorrent Server läuft.

3.0.2 Theoretische maximale Übertragungsrate der Systeme

Da in p2p Systemen die erste Quelle nur eine vollständige Kopie der Datei hochladen muss und danach die Knoten die Datei auch untereinander austauschen, wird die theoretische Übertragungsrate durch drei Effekte bestimmt:

- Die Einzelleistung der Knoten und vor allem der ersten Quelle.

  Es können nur diejenigen Dateifragmente verteilt werden, die die Quelle bereits hochgeladen hat.

- Die Effizienz des Algorithmus.

  In einer idealen Umgebung erhält von N Downloadern jeder einzelne Downloader 1/N-tel der Datei von der ersten Quelle und lädt den Rest der Datei von den anderen Downloadern.

  Dabei wird die Datei in Fragmente unterteilt, die deutlich kleiner als Dateigröße m / N sein sollten und einzeln verteilt werden, denn ein Downloader kann erst an der Verteilung teilnehmen, wenn er entweder von

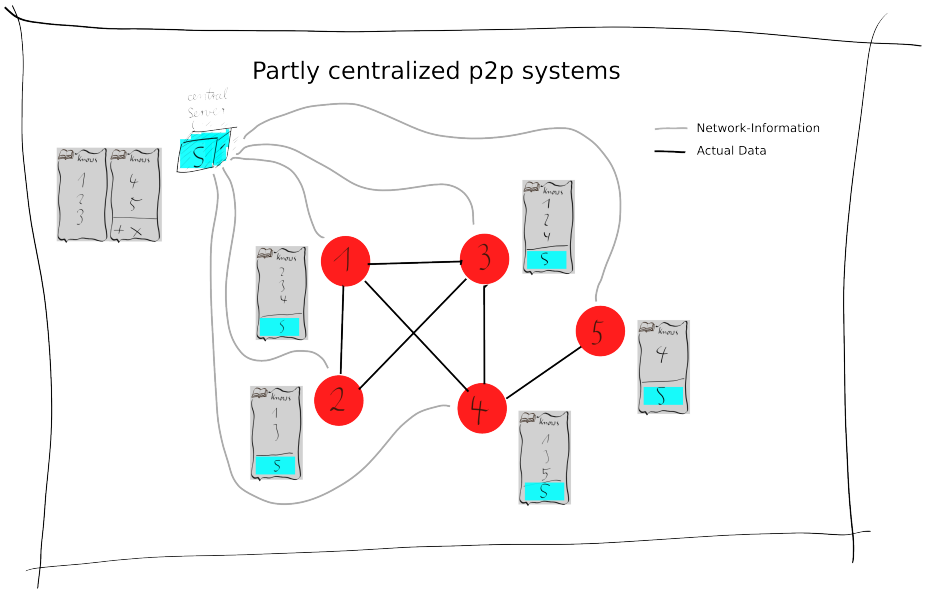


Abbildung 6: Struktur von teilzentralisierten p2p Netzwerken

der ersten Quelle oder von einem der anderen Downloader mindestens ein Fragment erhalten hat.

Je nach Algorithmus kann die hierfür benötigte Startup-Zeit höher oder niedriger sein, allerdings bei Knoten mit gleichen Einzelleistungen und ohne caching niemals niedriger als die benötigte Zeit in einem idealen Algorithmus, bei dem jeder Knoten sein Fragment sofort weitergibt, sobald er es erhalten hat:

$$log_2(N) * t$$

mit

$$t = Fragmentgroesse/Einzelleistung$$

- Die Gesamtleistung des Netzwerks.

  Die addierte Übertragungsgeschwindigkeit der einzelnen Knoten kann die Gesamtleistung des Netzes nicht übersteigen.

  In Clustern ist diese Gesamtleistung meist relativ fest begrenzt, während sie z.B. in WLAN-Meshes mit der Teilnehmerzahl steigt.

Dadurch ergeben sich drei Phasen für die Bestimmung der theoretischen Leistung:

1. Die Anfangsphase mit einer Dauer von mindestens

$$log_2(N) * Fragmentgroesse/Einzelgeschwindigkeit$$

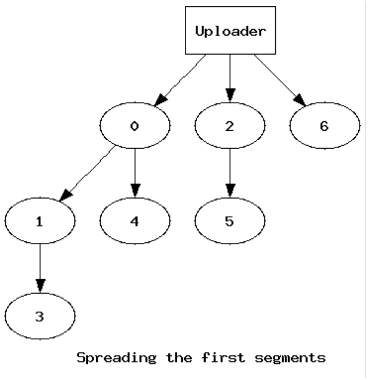
in der die maximale Gesamtgeschwindigkeit durch

$$\sum_{i=1}^{log_2(N)} (2^{i-1}/log_2(N)) * Einzelrate(upload)$$

gegeben ist (mit dem Maximalwert bei der Gesamtgeschwindigkeit des Netzes). Die Geschwindigkeit in der Startphase ist in 8 graphisch aufgetragen.

2. Die effekive Verteilphase, in der die maximal mögliche Geschwindigkeit gerade N*Einzelgeschwindigkeit ist, begrenzt durch die Maximalrate des Netzes.

3. Die Abschlussphase, in der bereits einige Downloader vollständige Daten haben. Hier ist die Geschwindigkeit v.a. durch die Akkumulierte Downloadrate aller Knoten mit noch unvollständigen Daten begrenzt. Deren Anzahl wird durch die Effizienz des Algoritmus gegeben. Im Idealfall sollte die Dauer der Anfangsphase nahe 0 sein, real ist sie deutlich größer.



Structure of the first fragments spreading in the network when a transfer gets initiated.

Abbildung 7: Anfangsphase: Die ersten Fragmente verbreiten

Gemeinsam ergibt sich also:
Übertragungsrate:

$$s_{ges} = \frac{Datenmenge}{t_{Anfang} + t_{verteil} + t_{Ende}}$$

Für unsere Umgebung ergibt sich also eine theoretische Maximalleistung von:
Einzelleistung: Festplattenleistung = 40-45MiB/s $\tilde{4}$2MiB/s
Maximalleistung Netzwerk = 1Gbps, gswitcht = 380 MiB/s
Fragmentgröße = 512kB

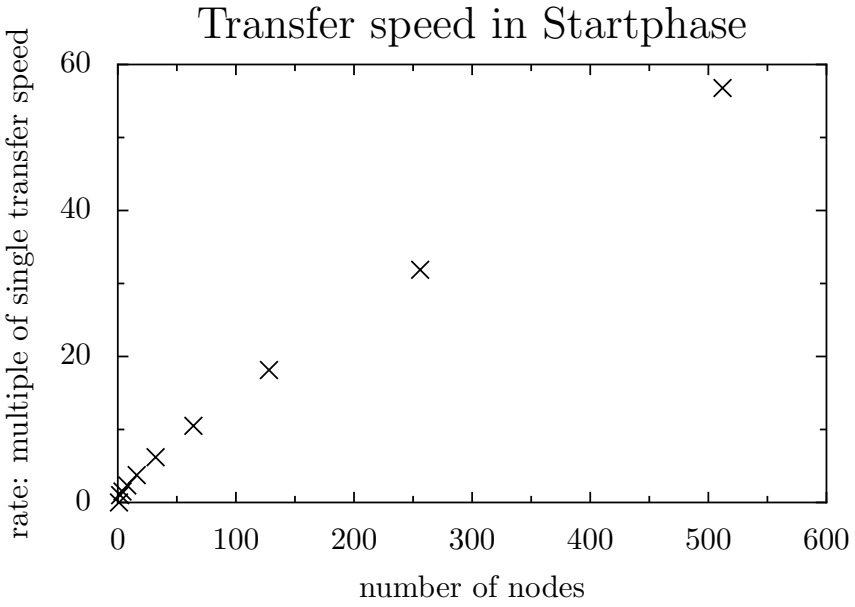


Abbildung 8: Effektive Übertragungsrate in der Anfangsphase bei Netzen mit unterschiedlicher Zahl von Knoten

Maximale Übertragungsrate:

$$s_{ges} = \frac{Datenmenge}{\frac{log_2(7)*Fragmentgroesse}{Plattenleistung)} + \frac{(Datenmenge - Anfangsrate*Anfangszeit)}{min(7*Plattenleistung, 380MiB/s)}}$$

Bei 816MiB also

$$s_{ges} = \frac{816MiB}{\frac{log_2(7)*512kiB}{42MiB/s)} + \frac{(816MiB - \sum_{i=1}^{log_2(7)}(2^{i-1}/log_2(7)) \ *log_2(7)*512kiB/42MiB/s)}{294MiB/s}}$$

Mit diskreten Werten ergibt sich:

$$s_{ges} = \frac{816MiB}{0.036s + \frac{816MiB - 0{,}083MiB}{294MiB/s}}$$

$$s_{ges} = 290.26MiB/s$$

Durch caching der Fragmente kann diese Geschwindigkeit noch weiter erhöht werden, da dann nicht die Platten, sondern die reine Netzwerkleistung der Knoten für die Berechnung der Einzelleistung aller Knoten bis auf die erste Quelle genommen werden kann.
Anschaulich bewirkt die Nutzung eines idealen p2p Systems, dass abgesehen von der Anfangs- und Endphase eine Übertragung von einem Knoten an alle Knoten in der Zeitspanne abläuft, die der eine Knoten für eine einzelne Übertragung

benötigt, solange die dabei insgesamt benötigte Bandbreite unterhalb der maximalen Bandbreite des Netzwerkes liegt, und dass ansonsten die Übertragung immer die volle Kapazität des Netzwerkes nutzt.
Mit dieser theoretischen Maximalgeschwindigkeit können wir nun verschiedene Techniken vergleichen.
Die einzelnen Abschnitte des Praktikums habe ich über Zielmarken aufgeteilt: Einzelne Schritte mit jeweils konkretem Ergebnis.

4 Ablauf und Ergebnisse

Einzelergebnisse der ersten drei Wochen.

4.1 Testmessungen und Installation

In der ersten Hälfte von Woche 1 habe ich Phex auf dem Cluster installiert und Geschwindigkeitstests durchgeführt. Der genutzte Cluster besteht aus einem Login-Knoten (Master) und 10 Arbeits-Knoten.
Dabei hat sich herausgestellt, dass Phex sehr stark auf Umgebungen optimiert ist, in denen die Netzwerkleistung der Flaschenhals ist. Auf Grund von Informationen des Hauptprogrammierers von Phex vermute ich, dass viele Buffer für LAN-Umgebungen zu klein eingestellt sind. Der Effekt war, dass es nur die halbe Netzwerkleistung nutzen konnte.
Auch wenn die Änderung der Buffergrößen technisch möglich gewesen wäre, hätte das den zeitlichen Rahmen dieses Praktikums gesprengt, so dass ich entschieden habe, mich bei den folgenden Messungen auf BitTorrent zu konzentrieren, da das der wahrscheinlichste Kandidat fure den Einsatz in Clustern ist. Auf einen Vergleich von Gnutella, edonkey und BitTorrent habe ich dadurch zu Gunsten eines stabilieren Tests von BitTorrent verzichtet.
Später habe ich dann herausgefunden, dass auch BitTorrent nur die halbe Leistung erreichte, so dass ein Test mit Phex sinnvoll vergleichbare Daten ergeben hätte.
Ich habe mich aber entschieden trotz allem bei einem reinen BitTorrent-Test zu bleiben, um stabilere Daten zu haben. Die Standardabweichung war schon so groß genug.
In der zweiten Hälfte von Woche 1 habe ich dann begonnen, eine BitTorrent-Testumgebung zu schreiben, um verschiedene Szenarien automatisiert testen zu können.
Dabei habe ich für den BitTorrent Server BitTornado verwendet, einen in Python implementierten BitTorrent Server und Client. Als BitTorrent Client habe ich rtorrent genutzt (in C++ geschrieben).
BitTornado habe ich verwendet, weil es damit sehr einfach ist, automatisiert Torrents zu erzeugen und freizugeben; die Grundfunktionen sind als einzelne Python Skripte verfügbar.
Rtorrent hat für diese Anwendung die Stärke, dass es vom Terminal aus genutzt wird (ncurses GUI) und mächtige Konfigurationsdateien hat, in denen sich z.B.

Grenzwerte für die genutzte Netzwerkgeschwindigkeit festlegen lassen, beides Stärken für den Einsatz in der automatisierten Testumgebung.
Beide Systeme sind freie Software, wobei rtorrent unter der GNU Public License v2 und BitTornado unter einer MIT-Lizenz verfügbar ist.
Ich habe einen BitTorrent-Server, einen Uploader und 7 Downloader genutzt, jede Datei also an 7 Knoten verteilt.
Von den Fragen konnte ich dabei die folgenden beantworten:

1. Welche Anforderungen haben die verschiedenen Systeme an Server und Clients?

   (a) Phex für Gnutella hat gleiche Ansprüche an Server und Client. Auf beiden muss Java installiert sein (>=1.5). Die Hauptlast für den Prozessor wird durch die TCP-Verbindung verursacht (Beobachtung der Ausgabe des Programmes top). Phex selbst verursacht nur sehr geringe Rechnerlast.

   (b) BitTorrent benötigt einen Informations-Server, die Download-Knoten und einen weiteren Knoten, der die Datei liefert (Uploader). Der Upload-Knoten kann zwar auf dem gleichen Rechner laufen, wie der Server, ich habe mich allerdings aufgrund der Netzwerkstruktur unseres Testclusters dafür entschieden, den Server auf einem eigenen Rechner zu starten: Auf dem Master.

   (c) Beide Systeme verursachten bei hohen Übertragungsgeschwindigkeiten durch die genutzten TCP Verbindungen eine hohe Prozessorlast.

2. Wie gut funktionieren die verschiedenen p2p Systeme in LAN-Umgebungen?

   (a) Phex für Gnutella hat durch die bereits beschriebene Optimierung einiger interner Buffer nur die halbe Einzelleistung geliefert.

   (b) Die gesamte von BitTorrent genutzte Bandbreite betrug nur ein gutes Viertel der möglichen Leistung des Netzes.

   (c) Beide Systeme haben Downloads nicht sofort gestartet, sondern hatten eine gewisse Startphase, in der keine Daten übertragen wurden (offset).

4.2 Testfälle (Variablen)

Nachdem ich die Testumgebung erstellt hatte, habe ich verschiedene Testfälle entworfen, an denen ich das System später getestet habe.
Die Testfälle waren als Szenarien in der Testumgebung erstellt, die einfach über die Befehlszeile aufgerufen werden können.
Ein Szenario ist dabei ein Python Script mit folgender Grundstruktur:

```
#!/usr/bin/env python
# encoding: utf-8
```

```
"""Distribute a file

Call this scenario via:

>>> scenario(run)

where ‘run’ is the function used to run the test.
"""

def scenario(run, logfile="logfile.log"):
    """Run the 'distribute many small files over WLAN' scenario."""

    #: Path to the source(s) to distribute.
    PATH = "/tmp/Infinite-Hands--free-software.ogg" # \~ 3.7MiB

    #: The number of individual tests to run.
    NUMBER_OF_TESTS = 30

    #: The time to wait for each test to complete.
    WAIT_TIME = 120 # 2 min

    #: The rtorrent config file for network speed throttling.
    CONFIG = "rtorrent-WLAN.rc"
    # this decreases the used network speed
    # to about 23Mbit/s - setting: \~ 2875kB/s, WLAN
    # which might be interesting for the OLPC project.

    # And now we just fire off the function which runs the test.
    # this is the only code we really need to copy in each file.

    run(path=PATH, number_of_tests=NUMBER_OF_TESTS,
        wait_time=WAIT_TIME, config=CONFIG, logfile=logfile)
```

Ich habe die folgenden Szenarien genutzt:

1. Mit unbegrenzter Netzwerkgeschwindigkeit eine große Datei (816MiB: Freier Film: Elephants Dream in High Definition Qualität). 10 Einzelmessungen.
2. Mit auf Modem-Geschwindigkeit begrenzter Netzwerkleistung und mittelgroßer Datei (24MiB: indymedia-g8_repression.ogg). 30 Einzelmessungen.
3. Mit auf WLAN-Geschwindigkeit begrenzten Verbindungen:
   (a) eine große Datei (816MiB: Elephants Dream HD). 10 Einzelmessungen.

(b) eine mittelgroße Datei (426 MiB: Elephants Dream 1024). 30 Einzelmessungen.

(c) eine recht kleine Datei (24 MiB: indymedia-g8_repression.ogg). 30 Einzelmessungen.

(d) eine kleine Datei (3,7 MiB: Infinite-Hands.ogg). 10 Einzelmessungen.

(e) 100 mal die kleine Datei. 60 Einzelmessungen.

Da unsere Platten auf /tmp nur begrenzten freien Platz hatten, konnte ich keine Tests mit sehr großen Dateien durchführen (z.B: mit DVD images). Für mögliche Nachfolgemessungen wären große Dateien v.a. bei unbeschränkter Netzwerkgeschwindigkeit interessant.
Die WLAN Messungen habe ich gewählt, weil ich für aussagekräftigere Messungen mit voller Leistung größere Dateien gebraucht hätte, weil mit diesen Ergebnissen Extrapolierungen für höhere Geschwindigkeiten möglich sein sollten, und weil sie für OLPC-XO Meshes interessant sein dürften. Beim letzten Punkt ist allerdings zu beachten, dass OLPC-Meshes 56 Mbit/s Netzwerkleistung bringen, ich aber mit WLAN 802.11 (23Mbit/s) getestet habe.
Die Modem-Messung sollte das untere Ende des Spektrums abdecken. Die Ergebnisse musste ich leider verwerfen, weil rtorrent hier die Bandbreitenbegrenzung nicht eingehalten zu haben scheint. Die Übertragungszeiten waren deutlich zu kurz.
Diese Schwierigkeit muss auch bei der Beurteilung der WLAN-Ergebnisse beachtet werden.

4.3 Die Messungen

Die genutzte Testumgebung wird im Anhang kurz umrissen.
Die Daten wurden erhoben, indem nach Abschluss aller Übertragungen die Erstellungszeit der ältesten Datei von der Änderungszeit der zuletzt modifizierten Datei abgezogen wurde. Nach dieser Zeit waren alle Übertragungen abgeschlossen.
Dabei ist nicht garantiert, dass jeder Download zu jeder Zeit aktiv war. Nach meinen Beobachtungen haben meist einzelne Clients den Download vor den anderen begonnen, und es gab einen festen Offset vor dem nur in den seltensten Fällen einer der Downloader mit der Übertragung anfing, obwohl die Dateien bereits angelegt waren.
Dieser Offset kann als Protokoll-Overhead angesehen werden, sollte jedoch bei größeren Clustern im Vergleich zur echten Übertragungszeit deutlich weniger wichtig werden.
Ich erhielt aus den Messungen die folgenden Ergebnisse:

4.3.1 Unbegrenzte Geschwindigkeit

Die erste Messung, unbegrenzte Geschwindigkeit und eine 816 MiB Datei, die von einem Uploader an 7 Downloader übertragen wurde, ergab eine Geschwindigkeit von (27.3 ± 2.5) MiB/s (siehe Abbildung 3), während eine vergleichende

Testmessung, bei der Daten mit scp übertragen wurden, ergab, dass Einzelübertragungen zu jedem Knoten eine Geschwindigkeit von etwa 21.5 MiB/s erreicht hätten.
Damit war hier die Übertragung mit BitTorrent um 27% schneller als einzelne Übertragungen. Die theoretisch zu erwartende Geschwindigkeit bei perfekt funktionierender verteilter Übertragung würde fast die 7-fache Geschwindigkeit der Einzelübertragung erreichen, bis fast zur Obergrenze der gesamten Netzwerkkapazität; bei uns (geswitchter 1000Mib/s LAN, Plattenleistung 42 MiB/s, 7 Knoten) 294MiB/s (genauer: 290MiB/s, siehe 2.2.2).
Die Verzögerung beim Starten der BitTorrent Clients kann aus den WLAN-Messungen abgeschätzt werden. Mit dem dort errechneten Offset von 132s kommt diese Messung auf effektiv (74.2 ± 18.3) MiB/s, was etwa 245% schneller ist, als die vergleichende Messung mit scp, immernoch aber nur 25.6% der gesamten Netzwerkkapazität von 290 MiB/s erreicht, was niedrig, aber nicht mehr völlig verheerend ist.
Bei deutlich mehr als 7 Knoten würde dieser Offset einen immer geringeren Anteil der Gesamtgeschwindigkeit ausmachen. Eine interessante weiterführende Messung wäre daher, wie sich der Offset bei der Nutzung von deutlich mehr Knoten verhält.
Ob der Offset aus dem WLAN-Test bei der Messung ohne Geschwindigkeitsbeschränkung genutzt werden kann ist allerdings nicht völlig sicher, daher liegt die wirkliche Beschleunigung wohl bei einem Faktor zwischen 1.27 und 3.45. Eine Messung mit unbegrenzter Geschwindigkeit und unterschiedlichen Dateigrößen könnte den hier auftretenden Offset prüfen.
Beide Daten, direkt gemessen und offset-korrigiert, sind in 9 aufgetragen.

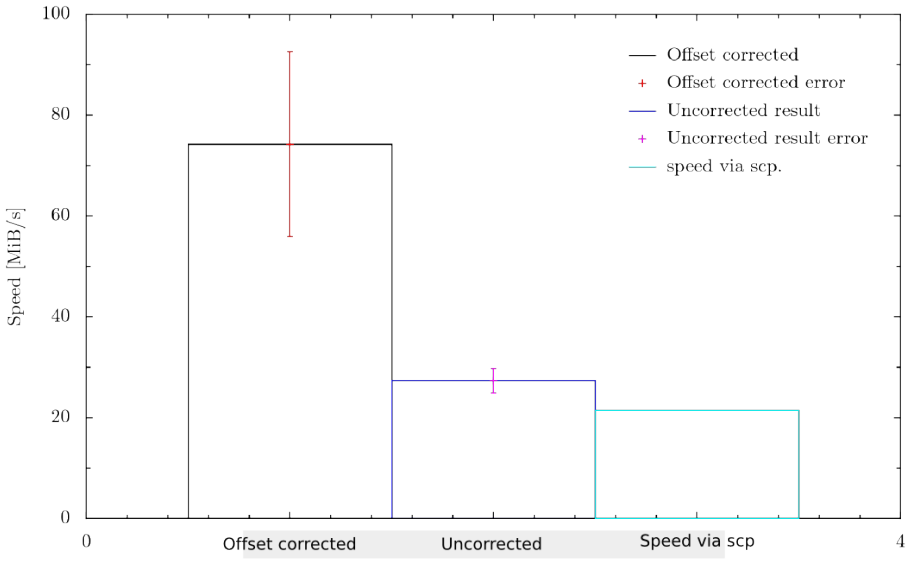


Abbildung 9: Geschwindigkeit der Übertragung bei unbegrenzter Bandbreite

4.3.2 56kbit/s - Modem

Die Modem-Messung musste ich leider verwerfen, da sie bei weitem zu hohe Übertragungsraten ergab. Ich vermute, dass die Bandbreitenbegrenzung in rtorrent bei den extrem niedrigen Grenzen nicht gegriffen hat.

4.3.3 23 Mib/s - WLAN

Für die nächste Messung wurde die maximale Geschwindigkeit jedes Clients auf 22.46 Mib/s = 2.808 MiB/s begrenzt, um eine WLAN-Umgebung zu simulieren.
Bei den kleineren Dateigrößen (24MiB und 3.7MiB) war die Geschwindigkeit niedriger als sie bei 7 Einzelübertragungen gewesen wäre.
Bei höheren Dateigrößen wurden Übertragungsraten zwischen 5.4 MiB/s und 7.9 MiB/s erreicht, wobei überraschenderweise die Übertragung der 100 kleinen Dateien die beste Leistung erreichte (10)
Diese Ergebnisse erreichten damit nur etwa die 2,0 bis 2,8-fache Übertragungsrate der einfachst möglichen Einzelübertragung.
Beim manuellen Beobachten der Downloads habe ich bemerkt, dass die Clients erst eine Datei anlegten und dann für einige Zeit warteten, bevor sie den wirklichen Download starteten.
Um die effektive Übertragungsgeschwindigkeit zu finden, und um abzuschätzen, wie sich die Leistung in größeren Clustern verhalten würde, habe ich daher die Übertragungszeiten aufgetragen (11) und linear gefittet. Daraus erhielt ich einen Offset von über 2 Minuten, in denen die Clients einfach warteten (132.358s, aus gnuplot, linear fit).
In 12 sind die Werte daher zur Prüfung der rohen Effizienz von BitTorrent offsetkorrigiert aufgetragen. In Abbildung 6 sind diese offset-korrigierten Werte relativ zur Übertragungsrate mit Einzelübertragungen aufgetragen, d.h. 13 stellt den Speedup dar.
Dass die Werte in der Messung mit extrem kleinen Dateien selbst offset-korrigiert unter der Geschwindigkeit von Einzeltransfers liegen, deutet darauf hin, dass entweder der Offset noch zu klein ist oder dass BitTorrent die Downloads nicht mit komplett festem offset startet, sondern jeder Client eine zufällige Verzögerung hat, die teilweise länger sein könnte als die Zeit um die Datei zu übertragen.
In 14 habe ich die Übertragungszeit mit BitTorrent nocheinmal linear gefittet und dazu die effektive Übertragungsgeschwindigkeit bei Einzelübertragungen aufgetragen.
An dem Vergleich wird deutlich, dass BitTorrent bei WLAN Geschwindigkeiten mit 7 Downloadern ab etwa 100MiB zu übertragenden Daten effizienter wird als Einzelübertragungen (hier ist der Offset noch nicht korrigiert, da die Offset Korrektur nur zur Abschätzung der Effizienz von BitTorrent in größeren Clustern dient).

4.3.4 Fazit

BitTorrent erreicht bei Datenmengen über 100MiB eine höhere Effizienz als einfache Übertragungen.

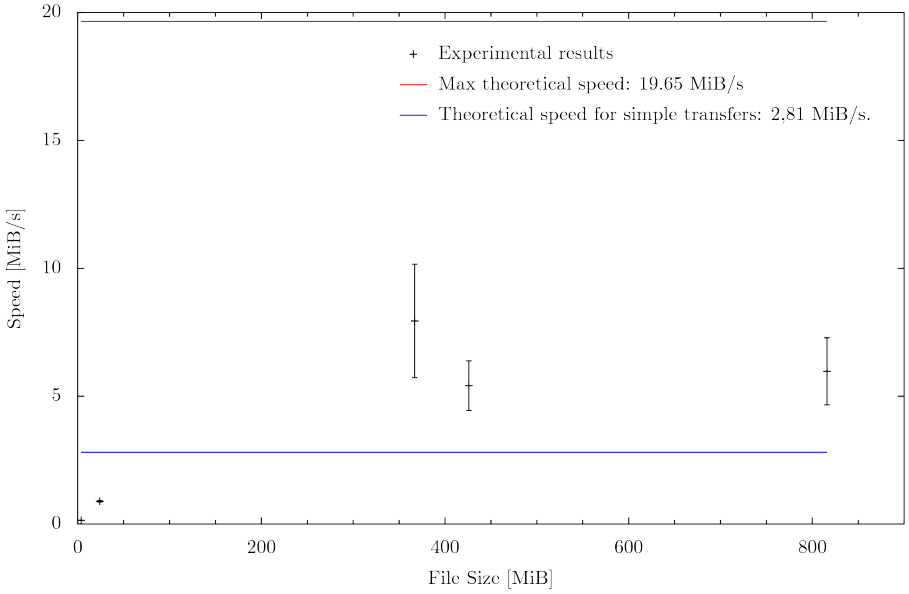


Abbildung 10: Geschwindigkeit der Übertragung bei WLAN Bandbreite

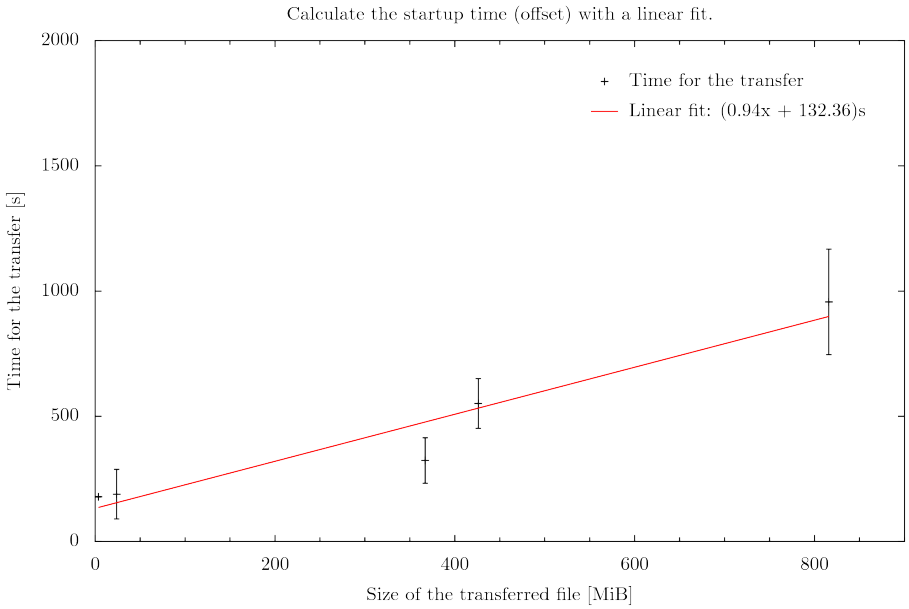


Abbildung 11: Offset-Bestimmung Übertragungszeiten - WLAN

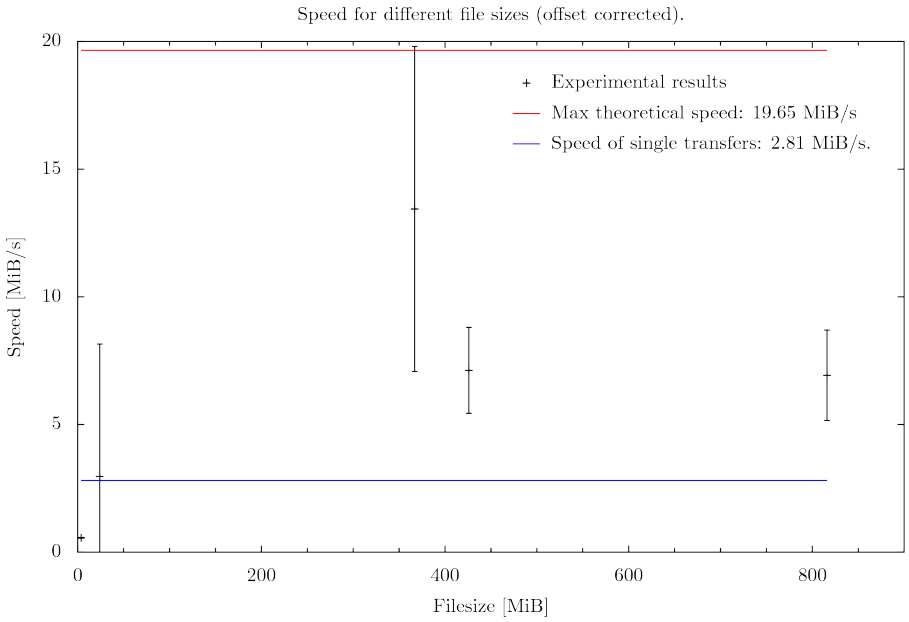


Abbildung 12: Offset-korrigierte Geschwindigkeit der Übertragung bei WLAN Bandbreite

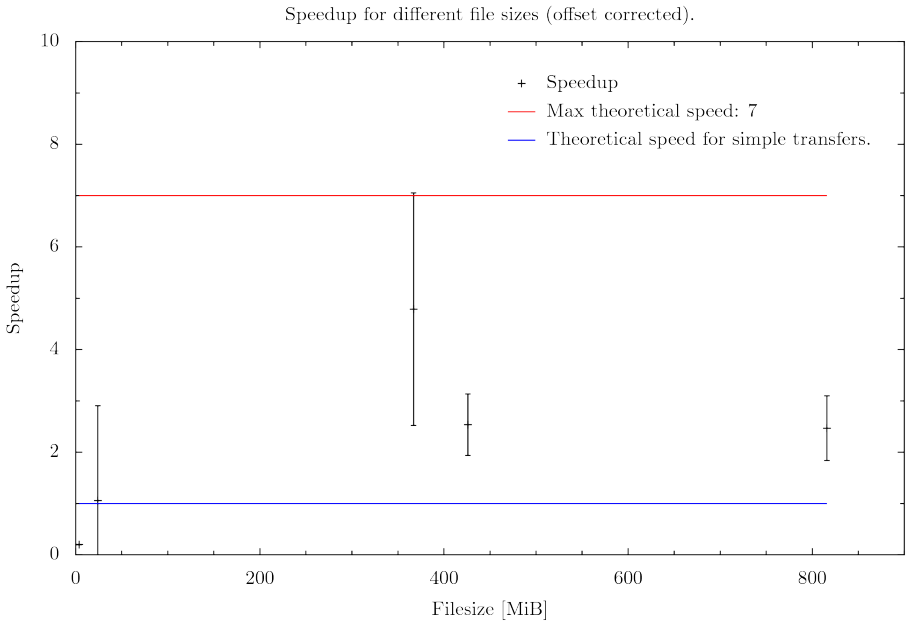


Abbildung 13: Offset-korrigierter Speedup der Übertragung bei WLAN Bandbreite

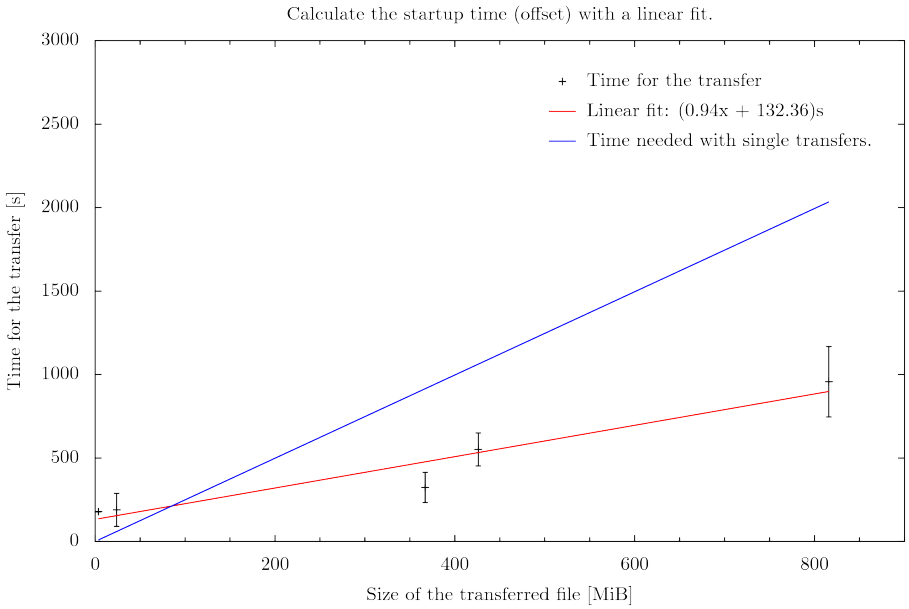


Abbildung 14: Vergleich der benötigten Zeit mit Einzelübertragungen und mit BitTorrent ohne Offset-Korrektur

In den hier getesteten Fällen erreichte es eine zwei- bis dreimal so hohe Übertragungsgeschwindigkeit.
In dieser Übertragungsgeschwindigkeit ist allerdings ein Offset enthalten, der wie bereits beschrieben, wohl größtenteils daher kommt, dass die Clients nach anlegen der Download-Datei kurz inaktiv werden, bevor sie die Übertragung beginnen.
Dieser Offset sollte in Setups mit deutlich mehr als 7 Downloadern unwichtig werden.
Eine wichtige Frage ist bisher allerdings ungeklärt: Warum war die Übertragung mit 100 gleichen Dateien so viel effizienter als die anderen Übertragungen? Um diese Frage zu beantworten wäre ein Test mit hundert unterschiedlichen Dateien interessant, um zu prüfen, ob die bessere Leistung von der Ähnlichkeit der Dateien oder von den vielen einzelnen enthaltenen Dateien herrührte.
Anders gefragt: Würde eine Übertragung von 100 unterschiedlichen Dateien ähnlicher Größe die gleiche Leistung erreichen?
Da die Abweichung vom Erwartungswert aber immernoch unter 2 Standardabweichungen der Messung liegt ($\tilde{9}5{,}4\%$ der Messungen sollten in 2 Standardabweichungen liegen), kann das Ergebnis auch einfach ein zufälliges Artefakt sein.

5 Auswertung und Endergebnis

Nachdem ich nun die Ergebnisse vorgestellt habe, möchte ich die grundlegende, in diesem Praktikum gestellte Frage beantworten, soweit es mir meine Messungen erlauben:

5.1 Ab welcher zu übertragender Datenmenge lohnt sich der Einsatz von BitTorrent in Clustern und anderen verteilten Systemen

Im Vergleich zu Einzelübertragungen, bei denen die Daten direkt von einem einzelnen Dateiserver an jeden der Zielrechner geschickt werden, lohnt sich BitTorrent in einem Setup mit 7 Zielrechnern und einem Sender und mit auf 23Mib/s begrenzter Einzelgeschwindigkeit (z.B. einem WLAN-Mesh, wie er von den OLPC laptops (XOs) aufgebaut wird) ab einer Datenmenge von etwa 100MiB, die auf jeden Knoten repliziert werden sollen.
Bei 816MiB übertragenen Daten erreichte die Übertragung die 2-3 fache Geschwindigkeit der Einzelübertragungen (siehe 9).
Da die Daten andeuten, dass in diesem Ergebnis ein fester Offset von etwa 2 Minuten enthalten ist, in dem die Rechner sich koordinieren, dürfte die Leistung für größere Verbünde deutlich steigen.
Durch Korrektur des Offsets konnte ich abschätzen, dass eine Übertragung mit BitTorrent in größeren Verbünden die 2,4 bis 4,8-fache Geschwindigkeit der Einzelübertragungen erreichen dürfte (siehe 9, offset-korrigierter Balken).
Bei einer unbeschränkten Verbindung über Gigabit Ethernet hat der Offset auch bei höheren Dateigrößen noch einen deutlichen Anteil an der Übertragungszeit.
In unserem Aufbau mit 7 Downloadern erreichte BitTorrent nur die 1,27-fache Geschwindigkeit von Einzelübertragungen, offset korrigiert stieg die Geschwindigkeit auf das 3.45-fache der Einzelübertragungen.
Selbst nach herausgerechnetem Offset wurde damit allerdings höchstens ein gutes Viertel der verfügbaren Netzwerkbandbreite genutzt.
Ein weiteres Ergebnis war, dass ein Großteil der Rechenzeit auf den Downloadern dabei mit dem Auspacken der TCP-Pakete verbracht wurde (Beobachtungen mit top). Durch die Nutzung von leichtgewichtigeren Protokollen wie UDP könnte daher die Leistung gesteigert werden.
In Clustern mit hinreichend schneller Verbindung (z.B. Gib-Ethernet) zeigt sich ein merklicher Nutzen von BitTorrent also erst bei sehr großen Dateien (über 800MiB) und/oder deutlich mehr als 7 Downloadern.
In WLAN Meshes, wie sie von den OLPC-Laptops (XOs) aufgebaut werden, wird er schon ab einer Datenmenge von 100MiB sichtbar, wie im breakeven Diagramm 14 aufgetragen.
Hier sehen wir in 13, dass für Dateien mit ausreichender Größe (über 200MiB) Speedups von 2.5 bis 5 erreicht werden konnten, was sich dem erwarteten Höchstwert eines Speedups von 7 annähert.

5.2 Wann können also p2p Techniken sinnvoll eingesetzt werden?

5.2.1 Cluster

p2p Techniken könnten in Clustern sinnvoll genutzt werden, in denen die folgenden Bedingungen gelten:

- bisher werden Daten entweder auf den einzelnen Rechnern gespeichert oder von einem einzelnen Dateiserver verwaltet, und
- jeder Rechner hat eine eigene Festplatte (oder kann die Daten vollständig im Speicher halten), und
- es müssen regelmäßig große Datenmengen an alle Knoten des Clusters verteilt werden, und
- sie verfügen über ein ausreichend schnelles Netz.

Ein ideales System würde dabei die Einzelgeschwindigkeit der ersten Quelle um einen Faktor gleich der Anzahl der Zielrechner erhöhen und durch caching im Speicher der Zielrechner könnte diese Geschwindigkeit noch weiter gesteigert werden, da dann die Leistung der Zielrechner nur noch durch ihre Netzanbindung begrenzt würde.
BitTorrent als spezifische Implementierung erzielt für Cluster jedoch keinen deutlichen Nutzen.
Es funktioniert allerdings sowohl mit einzelnen sehr großen als auch mit sehr vielen kleinen Dateien.
Wie in der Testumgebung realisiert, sind die Befehlszeilenwerkzeuge recht leicht mit Skripten zu steuern, und es kann die bestehende Infrastruktur des Clusters ohne grundlegende Änderungen genutzt werden.
Die Bedingung ist, dass vor der Verteilung alle Zielrechner bekannt und ein direkter automatisierter Zugriff auf sie möglich ist (z.B. über SSH). Außerdem brauchen alle Rechner in dem Cluster Zugriff auf den BitTorrent Server, und es muss möglich sein, einen BitTorrent Client zu installieren.

5.2.2 WLAN Meshes

Als weiteres Einsatzfeld von existierenden p2p Methoden zur Dateiverteilung bieten sich WLAN-Meshes an, da hier einfache Ansätze, wie eine strukturierte Übertragung von Rechner zu Rechner, durch die schnell wechselnde Topologie des Netzes behindert werden.
p2p Methoden wie BitTorrent bieten für diese Umgebung bereits Optimierungen, mit denen vor allem die nächsten Knoten genutzt werden (nämlich die, deren Geschwindigkeit am höchsten ist, die also über die wenigsten Zwischenschritte verbunden sind).
Außerdem sind in WLAN Meshes echte verteilte Dateisysteme deutlich schwerer zu realisieren.

BitTorrent zu nutzen kann dadurch, abgesehen von der sehr viel einfacheren Verwaltung und dem einfachen Initiieren der Übertragung, eine Leistungssteigerung auf das zwei- bis fünffache der Geschwindigkeit von direkten Einzelübertragungen erreichen, v.a. auch, weil hier jeder Knoten unabhängig von den anderen wirklich die volle WLAN-Leistung nutzen kann, so dass die Gesamtleistung mit der Anzahl der Knoten im Netz steigt, solange die Daten so oft wie möglich direkt übertragen werden.
In Netzen mit 7 Downloadern lohnt sich die Nutzung von BitTorrent durch den festen Offset ab einer Dateigröße um 100MiB, bei einer größeren Anzahl von Knoten vermutlich bereits bei kleineren Dateigrößen.
Allerdings könnte der zentralisierte BitTorrent-Server sich als Flaschenhals erweisen, da bei WLAN-meshes nicht bekannt ist, wie viele Knoten zwischen den einzelnen Rechnern liegen, so dass auch komplett dezentrale Methoden der Verteilung durch Gnutella Download-Meshes (HTTP basiertes, vollständig dezentrales swarming) geprüft werden sollten.

5.3 Alternativen und Verbesserungsmöglichkeiten

In Clustern sind Verbesserungsmöglichkeiten denkbar, die für Übertragungen im Internet noch wenig Sinn ergeben.

5.3.1 Verbesserungsmöglichkeiten für BitTorrent in Clustern

Bisher werden mit BitTorrent die Daten durch viele einzelne TCP-Verbindungen übertragen.
Um die Effizient für LAN-Netze zu erhöhen, könnte das BitTorrent Protokoll erweitert werden, um UDP-Broadcasts zu integrieren.
Dabei könnten am Anfang die Daten über einen UDP-Broadcast verteilt werden. Da UDP die Übertragung jedes einzelnen Fragments nicht sicherstellt, könnten sich danach die einzelnen Rechner fehlende Daten wie gewohnt über BitTorrent Verbindungen holen, so dass BitTorrent dezentral die Prüfung des UDP-Broadcasts übernehmen würde.
Natürlich könnten UDP-Broadcasts auch alleine eingesetzt werden, was jedoch nicht die Sicherheit von BitTorrent bietet, dass jeder Downloader die kompletten Daten hat, und auch nicht die Verteilung von Daten über das Internet oder in heterogenen Netzen ermöglichen würde.
Außerdem kann das Protokoll für eine Nutzung im Cluster je nach Anzahl von Knoten deutlich aggressiver auf den Server zugreifen, um die Dauer der Startphase zu minimieren.

5.4 Die Testumgebung

Für das Praktikum habe ich eine Testumgebung entwickelt, in der verschiedene Szenarien getestet werden können.
Sie ist in Python geschrieben und unter p2p performance testing in einem Mercurial Repository verfübgar.

Sie kann wie in 4.2 beschrieben genutzt werden um verschiedenste Szenarien zu testen, ist allerdings in ihrem aktuellen Status auf die Nutzung im kleinen Testcluster der Universität Heidelberg optimiert.