Newcastle University

COMPUTING SCIENCE

Query Matching in a BitTorrent-Based P2P Database System

Paul Watson and John Colquhoun

TECHNICAL REPORT SERIES

No. CS-TR-1184 **January 2010**

TECHNICAL REPORT SERIES

No. CS-TR-1184 January, 2010

Query Matching in a BitTorrent-Based P2P Database System

J Colquhoun, P Watson

Abstract

In our previous work,we introduced the Wigan Peer-to-Peer database server, which is based on the popular BitTorrent file-sharing protocol. In Wigan, users (peers) cache the results of queries they receive and make these available to future users. A central component, known as the Tracker, keeps a record of which users have submitted which queries and uses this record to provide a new user submitting a query with a list of one or more peers that already have these query results. In this paper, we describe the query matching process which occurs at the Tracker, thus highlighting the differences between query matching in a P2P database and file matching in a P2P file-sharing system and the challenges these differences posed.

Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

COLQUHOUN, J., WATSON, P.

Query Matching in a BitTorrent-Based P2P Database System
[By] J. Colquhoun, P. Watson

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2010.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1184)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-1184

Abstract

In our previous work ,we introduced the Wigan Peer-to-Peer database server, which is based on the popular BitTorrent file-sharing protocol. In Wigan, users (peers) cache the results of queries they receive and make these available to future users. A central component, known as the Tracker, keeps a record of which users have submitted which queries and uses this record to provide a new user submitting a query with a list of one or more peers that already have these query results. In this paper, we describe the query matching process which occurs at the Tracker, thus highlighting the differences between query matching in a P2P database and file matching in a P2P file-sharing system and the challenges these differences posed.

About the authors

Paul Watson is Professor of Computer Science, Director of the Informatics Research Institute, and Director of the North East Regional e-Science Centre. He also directs the UKRC Digital Economy Hub on "Inclusion through the Digital Economy". He graduated in 1983 with a BSc (I) in Computer Engineering from Manchester University, followed by a PhD in 1986. In the 80s, as a Lecturer at Manchester University, he was a designer of the Alvey Flagship and Esprit EDS systems. From 1990-5 he worked for ICL as a system designer of the Goldrush MegaServer parallel database server, which was released as a product in 1994. In August 1995 he moved to Newcastle University, where he has been an investigator on research projects worth over 20M. His research interests are in scalable information management. Paul Watson teaches information management on the System Design for Internet Applications Msc and the e-business MSc. In total, Paul Watson has over forty refereed publications, and three patents. Professor Watson is a Chartered Engineer, a Fellow of the British Computer Society, and a member of the UK Computing Research Committee.

On completion of John’s BSc in 2004, he began his PhD, supervised by Paul Watson. His research project concerned distributed query processing in peer-to-peer database systems. John submitted his Thesis in October 2007 and from then until December 2008, worked as an RA, still in the field of P2P database systems. Je was awarded my PhD in December 2008. In January 2009, he began work on the CVR (Cardiovascular Risk) project. He is writing software that will eventually be used by GPs to evaluate a patient's risk of cardiovascular disease and examine the effect of possible treatment options.

Suggested keywords

DATABASES
P2P COMPUTING

Query Matching in a BitTorrent-Based P2P Database System

John Colquhoun[1] and Paul Watson[1]

[1] School of Computing Science, Newcastle University, Newcastle-upon-Tyne, NE1 7RU, United Kingdom
{John.Colquhoun, Paul.Watson}@ncl.ac.uk

1. Introduction

In our previous work [1, 2, 3], we introduced the Wigan Peer-to-Peer database server, which is based on the popular BitTorrent file-sharing protocol [4]. In Wigan, users (peers) cache the results of queries they receive and make these available to future users. A central component, known as the Tracker, keeps a record of which users have submitted which queries and uses this record to provide a new user submitting a query with a list of one or more peers that already have these query results.

The Tracker is one of the key components of Wigan and its algorithms must work correctly in order to make Wigan a scalable P2P database. All users submitting queries to Wigan must be informed about those peers that already possess the query results. Failure to provide a new user with correct information may result in that user receiving incomplete query results and may also lead to these incorrect results being shared with other peers. This paper focuses on the process of query matching within the Wigan system. This is considerably more complex than file matching in BitTorrent (and other file-sharing systems), which has only two possibilities: either two files match or they do not. However, in a database such as Wigan, there are many complications which must be addressed. Firstly, database queries expressed in SQL will include fields in the "SELECT" clause and probably conditions in a "WHERE" clause. All of these fields and conditions must be checked against each other during the matching process. Furthermore, it is possible that two queries do not match exactly, but that one could still contain the results of the other, i.e. one query is a subset of the other. Equally, it is possible that two queries which logically are an exact match are expressed differently in SQL, e.g. the conditions in the "WHERE" clause are listed in a different order. There are also concepts including joins and aggregations which have no equivalent in file-sharing. In this paper, we introduce these challenges and present our solutions to them.

The rest of this paper is structured as follows. Section 2 provides an overview of BitTorrent and Section 3 introduces the Wigan architecture. Section 4 describes in detail how queries are matched at the Tracker, whilst Section 5 presents a suggestion for further work and concludes this paper.

2 BitTorrent Overview

BitTorrent [4] is a hybrid P2P file-sharing protocol [5]. The process of receiving a file in BitTorrent is called "downloading" and the corresponding process of providing a file to other peers is called "uploading." Similarly, peers engaged in these activities are known as "uploaders" and "downloaders." Uploaders advertise the file(s) they have copies of through a central component called a "Tracker." The Tracker acts as a directory, keeping track of which peers are downloading and uploading which files. Any peer that is advertising a complete file is known as a "seed", whilst any peer that is still in the process of downloading is known as a "leecher." There must be at least one seed present to introduce a file into the system and to place the first advertisement at the Tracker.
To start a download, a BitTorrent client will contact the Tracker and announce its interest in the file. Large files in BitTorrent are split into pieces, normally 256KB in size. The Tracker will provide a list of typically 50 random peers that already have some, or all, of the pieces. The downloader normally chooses the first piece at random and subsequent pieces in a rarest-first order. This allows rare pieces to spread further around the network. Once a downloader has received a complete piece, it is able to start uploading that piece to other downloaders. Thus, a BitTorrent leecher may be downloading and uploading different pieces of a particular file at the same time. A peer normally uploads to no more than five downloaders at any one time.

However, there are some peers that will operate according to a slightly amended lifecycle and will download but perform no uploading at all. These peers are called "Free Riders" and cause problems in BitTorrent and other file-sharing protocols because they consume resources but do not provide anything to other peers in return. BitTorrent's attempt to overcome this problem is to use a choking algorithm. "Choking" is the temporary refusal to upload a piece of a file to a particular downloader. The purpose of the choking algorithm is to ensure that those who provide little content into the system receive little in return.

3 Wigan Architecture

The Wigan system is derived from BitTorrent and hence the three major components in Wigan have the same names and basic roles as their counterparts in BitTorrent – the Seed, the Peers and the Tracker. Each is now discussed in turn.

3.1 The Seed

A Wigan seed possesses a complete copy of the database. Initially, the seed answers all queries (acting as if it were the server in a traditional client-server database). Once peers have begun to receive the results of queries; then they advertise them at the Tracker and can then answer each other's queries where possible as described below. However, if at any time a downloading peer submits a query which cannot be answered by any of the other available peers, the query is answered by the seed.

3.2 The Peers

The peers are the equivalent of clients in traditional client-server systems – they send out queries and receive the results. However, they also cache the results of the queries in a local database server. This allows them to answer each other's queries, so taking the load away from the seed and providing greater scalability. The way in which they do this is governed by the Tracker (described below). As in BitTorrent, there is no assumption made about the amount of time the peers spend connected to the system – a peer may decide to disconnect at any time.

3.3 The Tracker

The central component in the Wigan system is the Tracker. There is one Tracker per database and this performs the same basic functionality as its namesake in BitTorrent in that it provides the downloading peers with a list of possible uploaders for the query they are requesting. However, due to the increased complexity of database queries when compared to file access, the Wigan Tracker has much more functionality and complexity. Section 4 will describe the matching process in detail, whilst this section will introduce the basic functionalities of the Tracker.
When a peer issues a query, it is sent first to the Tracker. This holds information on all the queries that have already been executed, along with the id of the peer that is caching the result. These "adverts" are stored in a canonical form representing the tables, columns and conditions on these columns for each query.
When a query arrives at the Tracker from a peer, it checks these adverts to see which other peers could answer the query. In Wigan, it is possible for a downloader's query to match exactly with an advertisement. In this, it is similar to Memcached [6]. However, a key difference is that Wigan goes beyond this and supports answering queries that are a proper subset of one or more advertisements. An example would be:

Query: `SELECT item FROM parts WHERE cost <= 10`
Advert1: `SELECT item FROM parts WHERE cost <= 10`
Advert2: `SELECT item FROM parts WHERE cost <= 15`

The Wigan tracker would consider both of these advertisements as will be discussed in Section 4.

In addition to matching queries to advertisements, the Wigan Tracker performs other advertisement management functions. When a peer receives the results of a query, it will contact the Tracker stating

the query and the number of tuples it has received. This information is parsed into the Tracker's canonical form and is stored in the Tracker's database for use by a future downloader.

Uploading peers that wish to disconnect from the Wigan system also contact the Tracker. On receipt of such a message, the Tracker will delete all advertisements placed by that uploader.

3.4 Downloading and Uploading

We now examine the process of downloading and uploading. A new downloader must contact the Tracker with the SQL query that it wishes to execute. The Tracker will return a list of suitable adverts grouped by query (more details of this will be discussed in Section 4) and the downloader must first select a group. For performance reasons, the downloader will choose those queries which exactly match the one it is searching for if this is possible or if it is not, start with the closest to an exact match. Note that "closeness" is determined by the number of tuples. A downloader will select uploaders advertising the query with the smallest number of tuples. This is also for performance reasons – uploaders which have a larger result set to scan may take longer to return the results.

The downloader contacts a randomly selected uploader peer from its chosen query group and submits a query for the first piece. If the uploader is able to accommodate a new downloader, it will perform the query and return all tuples from the first piece which matches the conditions of the query. A header with the query, piece number and a query ID is included so that if a downloader is receiving multiple queries simultaneously it can correlate responses to requests. Note that if there is no data in the first piece which matches the conditions of the query, the uploader will still send a response, containing just the header and no tuples. This prevents the downloader from assuming the response has gone missing because of a technical problem.

Once the first piece has arrived, the downloader stores the data in its local database and then makes a request for the next piece (potentially to a different peer). This process continues until the downloader has received all of the pieces. The downloader knows when this point occurs because the Tracker has informed it of the number of pieces. To improve performance, query requests for different pieces can be sent to a set of peers in parallel.

A BitTorrent peer can begin uploading as soon as it receives a complete piece of the file. However, a Wigan uploader cannot do this because it may be receiving data from an uploader advertising a different query. If the downloader has received its data from a peer advertising a different query, it will have to change the piece structure before it begins to upload.

For example, consider a university department's database, which has a Student table containing details of all students studying in the department. This table is split into 20 pieces. Initially, there is one seed containing the whole database and therefore a complete copy of the Student table. A new downloader requests the following query:

```
SELECT * FROM student WHERE tutor = 'Lee'
```

During the download, the downloader will send 20 requests, one for each piece. For each request, the seed will send data from that piece containing details of all students whose tutor is Lee. Let us assume that there are 20 such students. There is no guarantee of how these 20 students' details are distributed across the pieces. They may all be stored in one piece, in which case the downloader will receive 19 empty responses. This happens because the downloader has to request data from each piece; it does not know in advance which pieces will contain data matching the query. At this point there is no alternative option because the downloader has to query all 20 pieces. However, when the downloader makes this data available to others, it would not make sense from an efficiency point of view to have 20 pieces again. Instead, these resulting 20 tuples could all be grouped together in the minimum number of pieces.

Once a peer has made any required changes to the piece structure, it contacts the Tracker, stating it has received the query and informing it of how many tuples it received. The peer's advert is then stored at the Tracker and the peer becomes an uploader. Whilst uploading, it may receive requests periodically from downloading peers asking it to provide data. There is no specific amount of time that a peer has to upload for, as in BitTorrent, a peer can disconnect at any time.

4. Query Matching at the Tracker

Section 3.3 noted that the Wigan Tracker has to perform more functionalities that its counterpart in BitTorrent. The BitTorrent Tracker simply has to look for those peers which have a full or partial copy of a file, pick up to 50 at random and return a list of these peers to the downloader as described in Section 2. Any uploader with a piece of the file will suffice. With regard to matching the adverts to requests, there are only two possibilities – either they match or they do not.

In contrast, in Wigan, the conditions in the "WHERE" clause of a query mean that it is possible that one query over a particular table cannot be resolved by an advert for another query over the same table. The Tracker must filter out these unsuitable adverts and return only those that can resolve the downloader's query. There are four possible relationships between the query and the advert which will now be described.

The first case is that the advert and the query do not match each other at all as shown in Figure 1 below.

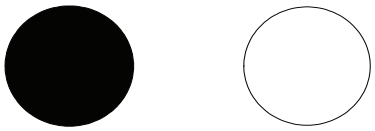

Figure 1 – A query (white circle) and an advert (black circle) which do not match

In SQL, an example would be:

Query: "`SELECT item FROM parts WHERE cost <= 10`"

Advert: "`SELECT item FROM parts WHERE cost >= 50`"

These adverts are of no use to the downloader and thus the Tracker must not inform the downloader about such adverts.

The second case is that the query is a partial subset of the advert as shown in Figure 2 below.

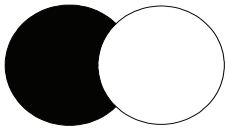

Figure 2 – A query (white circle) which is a partial subset of an advert (black circle)

In SQL, an example would be:

Query: "`SELECT item FROM parts WHERE cost <= 10`"

Advert: "`SELECT item FROM parts WHERE cost <= 5`"

Such an advert on its own cannot resolve the downloader's query. In the example SQL above, the advert does not provide any information about parts which cost between £5.01 and £10. This information would have to be obtained from another advert. The current Wigan Tracker would not return adverts like this which could not resolve the query completely. A future extension to Wigan would be to return a collection of partial subsets so that the downloader's query can still be resolved by obtaining different parts of the query from different adverts.

The third case is that the query is a proper subset of the advert as shown in Figure 3 below.

Figure 3 – A query (white circle) which is a proper subset of an advert (black circle)

In SQL, an example would be:

Query: "`SELECT item FROM parts WHERE cost <= 10`"

Advert: "`SELECT item FROM parts WHERE cost <= 15`"

Although the query and advert are not identical, the advert can resolve the downloader's query. The seed is a special example of this case. The seed has the entire database hence any query executed on the database is a subset of the seed's data. It is perfectly acceptable for the Tracker to return such adverts, though an exact match is preferable if possible, for reasons that will be described below.

The final case is that the query and the advert are identical as shown in Figure 4.

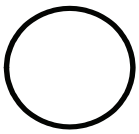

Figure 4 – A query and advert which are identical

In this case, there is an uploading peer advertising exactly the same query as is required by the downloader. An exact match is preferable to the previous example of a proper subset, shown in Figure 3, for both the uploader and the downloader. The uploading peer will have to perform fewer I/O's and no filtering. If the query is a proper subset of the advert, there is likely to be some data in the advert which the downloader does not require. The uploader, however, has to scan all of the data it holds to see if it should be returned to the downloader hence some data will be scanned but not selected. A particularly bad case of this occurs when a query which has a small result set is executed at the seed. An entire table may be scanned for only a few results to be obtained. Indexing may help overcome this problem. However, with an exact match, this issue will not happen as all data scanned is returned as part of the result set. From the downloader's point of view, it is likely that an exact match will also take less time to be returned.

Another major difference between the BitTorrent and Wigan Trackers is that, instead of receiving a request for a particular file, a Wigan downloader sends a database query. In the simulator and 'live' versions of Wigan [1, 2, 3], and also in the examples discussed in this paper, these queries are expressed in SQL. Although some existing P2P database systems, for example BioWired [7] and PIER [8, 9], have their own query processing languages, SQL provides all of the required query processing expressions and also offers scope for more complex queries, for example nested subqueries. It is also a common, standardised language known to many developers and used in many applications. This may allow future versions of Wigan to integrate with existing database applications. Note that it should also be possible to implement the Wigan architecture using other existing query languages if required.

The Tracker holds all the uploaders' advertisements, in a database using a canonical form representing the table names, *columns* from the tables and *conditions* on these *columns* (the elements contained within a "WHERE" clause in SQL) for each query. This allows for SQL queries written in any order, for example in the Tracker's canonical form, the query "`SELECT name, address, phone FROM person`" will be identical to the query "`SELECT name, phone, address FROM person`". Note that in this paper, to distinguish between the columns in the Tracker's database tables (and the conditions on the data they hold) and those in the database tables being advertised, those in the tables being advertised will be shown in *italics*.

When the Tracker receives a downloader's query, it must examine the adverts in its database and compare these to the query in order to find relevant adverts. The Tracker's database contains two tables, one of which contains data about the *columns* being advertised and the other containing details of the *conditions* on these *columns*. Two examples of this matching process will now be described.

Consider a database containing three tables, T1, T2 and T3. The following example illustrates a scenario where a peer, Peer A has just completed the download of the query "`SELECT x, y FROM T1 WHERE y < 100`". Peer A is the first peer to finish downloading a query; hence the Tracker contains details of only the seed's advert and Peer A's advert at this point. Figure 5 shows the resulting Tracker database table for *Columns*.

mappingId	peerId	tableName	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11	cc
1	Seed	T1	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YE
2	Seed	T2	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YE
3	Seed	T3	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YE
4	Peer A	T1	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO

Figure 5 – the Tracker's database for columns information, with only the seed and a single peer advertising

The first column is a mapping ID and is used by the Tracker to join rows from the *Columns* and *Conditions* tables when matching queries to adverts. The second column shows the ID of the peer placing the advertisement and the third column shows the table the advertisement covers. The remaining columns show, for each *column* in the table being advertised, if that advert contains those *columns* or not. For example, Peer A is shown as having only two *columns* from the table T1, but the seed has all *columns* from all tables. If a peer had performed an aggregation on a *column*, for example "MAX", this would be shown instead of "YES". An example of this will be shown later in this section.

Figure 6 shows the Tracker database table for *Conditions*.

mappingId	peerId	tableName	dataSize	col1Condition	col2Condition	col3Condition	col4Condition
1	Seed	T1	1000	NONE	NONE	NONE	NONE
2	Seed	T2	100	NONE	NONE	NONE	NONE
3	Seed	T3	50	NONE	NONE	NONE	NONE
4	Peer A	T1	100	NONE	< 100	NONE	NONE

Figure 6 – the Tracker's database for conditions with only the seed and a single peer advertising

The first column is a mapping ID which corresponds to the mapping ID in the *Columns* table. The peer ID and the table name are the next two columns. The fourth column shows the number of tuples received by the uploading peer. The remaining columns show *conditions* on *columns* in the table being advertised. For example, Peer A has no *condition* on the first *column* (x), but there is a *condition* on the second *column*, because the "WHERE" clause of Peer A's query contains "y < 100". The seed has no *conditions* on any *columns*, because it holds all data from all tables. Note that if a query contained a join, an entry for each of the tables in the "FROM" clause would be made in both of the Tracker's database tables. The join *condition (s)* (e.g. "`= T2.key`") would be shown in the Tracker's table of *Conditions.*

Now consider a new downloader, Peer B, which submits the query "`SELECT x, y FROM T1 WHERE y < 10`" to the Tracker. This query is a proper subset of Peer A's. The Tracker initially retrieves all peers which have the correct *columns* from the relevant table, T1 in this example. This retrieval process will return both the seed and Peer A as each possesses data from the first two *columns*, as shown in Figure 7.

peerId	datasize	col1condition	col2condition	col3condition	col4condition	col5condition	col6condition
Seed	1000	NONE	NONE	NONE	NONE	NONE	NONE
Peer A	100	NONE	< 100	NONE	NONE	NONE	NONE

Figure 7 – the result of the initial selection process

The Tracker then checks the *conditions* on the relevant *columns* for these advertisements. For the first *column* (x), the seed has no *conditions* and neither does Peer B's query. For the second *column* (y), the seed also has no *conditions* and Peer B's query has a *condition* (y < 10). The seed's data will include items where y < 10, which is acceptable. The Tracker must also ensure there are no *conditions* on other *columns* which will restrict the result set, for example an advert for
"`SELECT x, y FROM T1 WHERE x < 5 AND z < 10`" could not resolve the query
"`SELECT x, y FROM T1 WHERE x < 5`"
because the *condition* "z < 10" further restricts the result set and will not necessarily return all items where x < 5. There are no such additional *conditions* here however, thus the seed's advert is accepted.

The Tracker then examines Peer A's advertisement. Like the seed, there is no *condition* on the first *column*. However, there is a *condition* (< 100) on the second *column*. These *conditions* are checked against each other. Given that anything that is less than 10 will be included in a results set containing data less than 100 and Peer A has no other *columns* which could affect the result set, this advert is also accepted. Once both relevant adverts have been checked, the selection process terminates. In this example, the Tracker has found two adverts that could satisfy the downloader's query. Note that both of these adverts are for different queries, neither of which was that requested by the downloader.

The Tracker groups the adverts by query. For each group, it lists the number of tuples – and also the number of pieces – in addition to the ids of the peers with that query. These are ordered based on a priority system, which favours adverts matching the downloader's query for the reasons explained earlier. If, as in this example, none of the adverts match the downloader's query, the highest priority is given to the closest to an exact match. The closest is given in terms of the number of tuples in the query results set. In this example, Peer A has 100 tuples but the seed has 1,000 and hence Peer A is the closest to an exact match. Note that if there are multiple adverts that could answer the query, then the seed will always have the lowest priority as there will be another advert which will have a smaller results set to examine. For this example, if we assume 50 tuples per piece, the groups will be ((100, 2, Peer A), (1000, 20, Seed)). The Tracker then returns the groups to the downloader.

The example described above was fairly straightforward because there were only two adverts, both of which could satisfy the downloader's query. A more complex example will now be described. Consider the same database used in the previous example. Over time, more peers have downloaded queries and are advertising at the Tracker, as shown in Figure 8 (*columns*) and Figure 9 (*conditions*).

mappingId	peerId	tableName	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11	c
1	Seed	T1	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	Y
2	Seed	T2	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	Y
3	Seed	T3	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	Y
4	Peer A	T1	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	N
5	Peer B	T1	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	N
6	Peer C	T1	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	N
7	Peer D	T1	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	N
8	Peer E	T1	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	N
9	Peer F	T1	YES	YES	YES	YES	NO	NO	NO	NO	NO	NO	NO	N
10	Peer G	T1	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	N
11	Peer H	T2	YES	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	N
12	Peer I	T1	YES	MAX	NO	NO	NO	NO	NO	NO	NO	NO	NO	N

Figure 8 – the Tracker's database for columns later in the download period

Note that the final advert, placed by Peer I in Figure 8 is for a query which includes an aggregation ("MAX") on *column* 2.

mappingId	peerId	tableName	dataSize	col1Condition	col2Condition	col3Condition	col4Condition
1	Seed	T1	1000	NONE	NONE	NONE	NONE
2	Seed	T2	100	NONE	NONE	NONE	NONE
3	Seed	T3	50	NONE	NONE	NONE	NONE
4	Peer A	T1	100	NONE	< 100	NONE	NONE
5	Peer B	T1	10	NONE	< 10	NONE	NONE
6	Peer C	T1	10	NONE	< 10	NONE	NONE
7	Peer D	T1	10	NONE	< 10	NONE	NONE
8	Peer E	T1	995	NONE	> 5	NONE	NONE
9	Peer F	T1	6	NONE	< 10	> 1 AND < 10	NONE
10	Peer G	T1	10	< 10	NONE	NONE	NONE
11	Peer H	T2	100	NONE	< 50	NONE	NONE
12	Peer I	T1	1	NONE	NONE	NONE	NONE

Figure 9 – the Tracker's database for conditions later in the download period

Note that the advert placed by Peer F (mapping ID 9) shows an example of a query where multiple *conditions* occur on the same column, in this case "> 1 AND < 10" in column 3.

A new downloader, Peer J, submits "`SELECT x, y FROM T1 WHERE y < 10`" to the Tracker. Once again, the Tracker will retrieve all peers with the correct *columns* from the relevant table. The results of this query are shown in Figure 10.

peerId	datasize	col1condition	col2condition	col3condition	col4condition	col5condition	col6condition
Seed	1000	NONE	NONE	NONE	NONE	NONE	NONE
Peer A	100	NONE	< 100	NONE	NONE	NONE	NONE
Peer B	10	NONE	< 10	NONE	NONE	NONE	NONE
Peer C	10	NONE	< 10	NONE	NONE	NONE	NONE
Peer D	10	NONE	< 10	NONE	NONE	NONE	NONE
Peer E	995	NONE	> 5	NONE	NONE	NONE	NONE
Peer F	6	NONE	< 10	> 1 AND < 10	NONE	NONE	NONE

Figure 10 – the result of the initial selection process

It can be seen that not all of the advertisements shown in Figure 8 and Figure 9 have been included. Peer G's advert was rejected because it has only one of the required *columns*, x. Peer H's advert was for a query on one of the other tables. Finally, the advert placed by Peer I was rejected because it contained an aggregation ("MAX") on the second *column*, y, and not the complete column data.

The Tracker will now examine the *conditions* in each of the advertisements to see if they can be used to satisfy the downloader's query. Again, the adverts placed by the seed and Peer A can be accepted, as discussed in the previous example.

The next three adverts, placed by Peers B, C and D can be accepted quickly because they contain identical *conditions* to the downloader's query – nothing on the first *column* and "< 10" on the second.

Peer E's advertisement is checked next. The *conditions* on the first *column* are acceptable (neither the query nor the advert has any), but those on the second *column* are not. The downloader is interested in items where y < 10, but Peer E is advertising data on items where y > 5, thus Peer E's advert is rejected.

The *conditions* on the first two *columns* in Peer F's advert are the same as those in Peer J's query. However, Peer F has a *condition* on another *column* ("> 1 AND < 10" on *column* 3) which means that this advert will not be able to answer the query. There may be some tuples which match Peer J's query that are not included in Peer F's results and hence this advert is rejected.

At this point, all the relevant adverts have been checked with those placed by the seed and Peers A, B, C and D being accepted. These adverts will now be placed in groups and returned to the downloader. The grouping process is the same as described in the previous example, however this time there are more groups because there are more uploaders with different queries. Assuming 50 tuples per piece again, the groups returned to Peer J are

((10, 1, Peer B, Peer C, Peer D), (100, 2, Peer A), (1000, 20, Seed)). It can be seen that the first group contains multiple uploaders. These peers are all advertising the same query and hence are all included in the one group.

Later in the download period, there may be a very large number of peers able to answer a new downloader's query. In BitTorrent, the Tracker selects 50 random peers' adverts to return if there are more than 50 suitable peers available, as described in Section 2. The Wigan Tracker follows a similar approach, however, the selection of peers is based on the query groups described above. The selection is chosen on the closeness of the advert to the downloader's query. Returning to the first example from this section, if there were another 49 adverts identical to Peer A's in addition to the seed's advert, there would be a choice – either return the seed's advert and 49 of the others, or return Peer A's advert and the other 49 identical adverts. Given that Peer A's advert is closer to the downloader's query than the seed's, this latter approach is taken.

Now, consider a scenario later in the download period where there are not yet 50 adverts for the downloader's query, but more than 50 other adverts which can satisfy the query. The Tracker examines the groups, starting with that which is furthest away from being an exact match. This is measured by the number of tuples as described earlier. If this entire group can be removed from the set of adverts found, still leaving 50 or more adverts, then it is. If it cannot, then adverts are removed at random until only 50 adverts remain in total. Note that adverts are only removed from this particular result set and are not deleted from the Tracker's list of advertisements. If there are 40 adverts for some query Q, 15 adverts for a query, Q1, such that $Q \subseteq Q1$ and the seed, the Tracker will return the 40 adverts for Q and a random selection of 10 adverts for Q1. This process is illustrated in Figure 11.

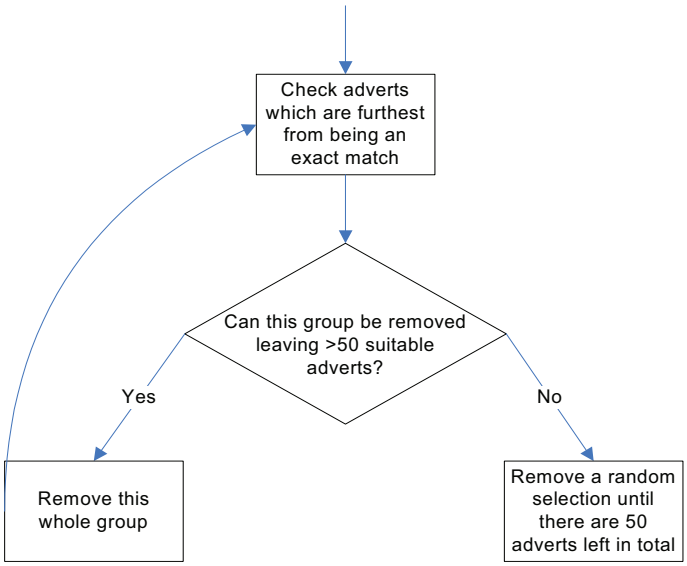

Figure 11 – Choosing 50 adverts from the selection.

Finally, it is important to note here that the Tracker is only responsible for managing adverts and matching them to queries. The Wigan Tracker does not contain any query results itself. Also, as in BitTorrent, the Tracker does not control the download process; this is left to the downloading peer. Whilst it is logical for peers to begin downloading from the closest possible group to an exact match, i.e. the first group in the list provided by the Tracker and thus the reason why the Tracker sorts the groups in the manner it does, the Tracker does not force peers to adhere to this policy. It does not prevent peers using other metrics to decide which group of peers to contact first, nor does it ever attempt to execute queries on the downloaders' behalf.

5 Conclusions and Further Work

The algorithms described in this paper are those currently utilised in our implementation of the Wigan system. One future extension to Wigan affects the matching process and Section 5.1 will describe this in more detail.

5.1 Partial Subsets

Section 4, and Figure 2 in particular, illustrated an example of a query which is a partial subset of an advertisement, i.e. that advertisement alone cannot satisfy the query, and noted that in the present Wigan implementation, such advertisements were not suggested to downloading peers. One extension therefore, would be for the Tracker to return partial subsets to the downloading peer. When examining which uploaders can provide each table, instead of rejecting peers that do not have all the correct tables, columns or rows, those which could resolve part of the query would be included in the result set. In the same way in which the Tracker currently arranges the peer list in groups, something similar must be done to ensure the downloader possesses enough information about where it can obtain the data. A downloader must be told by the Tracker exactly which uploaders possess which parts of the query. The Tracker must not provide one partial subset if the remainder of the query cannot be obtained from any other.

It may be viewed as unnecessarily complex to obtain a query as a collection of partial subsets if there is at least one uploader available that can provide the entire query. However, one scenario where partial subsets might be advantageous would be in the case of a query answerable only by the seed. If a downloader discovered that their query was answerable in its entirety by the seed or as a collection of partial subsets from other peers, the downloader may decide to choose the latter option. Not only would this reduce the load on the seed, but the downloader may experience a faster response time if the other peers were not as busy as the seed. Another scenario in which partial subsets could be of use is if the query has a potentially large result set and there are only a small number of possible uploaders which means that the advantages of sending piece requests in parallel cannot be exploited fully. In this case, concurrently obtaining parts of the data from partial subsets could be faster.

When examining the issue of partial subsets, one point of importance is how involved the Tracker should become in the download process. There will always be at least one peer that can resolve any query in its entirety. However, if there is only one suitable uploader (the seed), should the Tracker split up the query automatically into various parts and attempt to find additional peers that could resolve those parts? Alternatively, should it just return the one uploader and allow the downloader to resubmit the query if they wish? Note that this would involve the downloader's Wigan software making a dynamic decision to resubmit the query instead of accepting the peer list supplied by the Tracker and commencing the download. Clearly, this latter approach could inconvenience the downloader because it involves the downloader's software having to perform more tasks and re-contact the Tracker. A better option would be for the Tracker to attempt to find peers that could answer the query in its entirety, and if there was less than the maximum number of permitted uploaders on the peer list (50 in our implementation), fill the list with any collections of partial subsets. Clearly, this would require a more advanced Tracker, which would need to iterate through all available adverts with the correct columns to find out if any of these adverts could answer part of the query.

There is also the choice about how the Tracker groups the adverts. Consider the query "`SELECT item FROM parts WHERE cost >= 50 AND cost <= 100`". Assume that a downloader submits this query and that there are currently five advertisements at the Tracker which could answer all or part of the query:

1) `SELECT item FROM parts WHERE cost >= 75 AND cost <= 100`

2) `SELECT item FROM parts WHERE cost < 75`

3) `SELECT item FROM parts WHERE cost > 50 AND cost <= 100`

4) `SELECT item FROM parts WHERE cost <= 50`

5) `SELECT * FROM parts` (The seed)

The results could be obtained from combining the results of Adverts 1 and 2, by combining the results of Advert 3 and Advert 4 (to find the items which cost £50 exactly) or by using Advert 5. There are three possible ways of grouping the adverts in the peer list, each demonstrating a deeper level of involvement by the Tracker. Firstly, it could simply return a list of peers grouped by advert, stating what each group of peers was advertising. In this case, the downloader's Wigan software would calculate which combinations of adverts would provide the entire query results and pick an option to execute.

Alternatively, the Tracker could return adverts in a different grouping, so that each grouping would return the entire query. In the example here, the first group would contain all peers advertising Advert 5, the second would contain all those advertising Adverts 1 and 2 and the third would contain all those peers advertising Adverts 3 and 4. This also gives the downloader a choice of how to obtain the query, but saves the downloader some time because it does not have to work out how to combine the groups in order to obtain the results; instead it just selects its preferred option.

Finally, the Tracker could select which of the three groupings was the best option and only return that information to the downloader. For example, if the Tracker believed the seed was not very busy, it could only inform the downloader about Advert 5. This saves even more time at the downloader, but it does result in the Tracker making an important decision on the downloader's behalf. If the Tracker made an unsuitable decision, the downloader may experience a sub-optimal response time and would be unaware of potentially better options for obtaining the results. It could be argued that all decisions about how to download the query should be made by the downloader. The process of choosing a collection of partial subsets is, in effect, introducing costed query plans into the Wigan system, both at the Tracker and at the downloader.

Care must also be taken to avoid the Tracker becoming a bottleneck. If the Tracker has to make more decisions, it will take longer to process each request. If the system was very busy, queues may form at the Tracker if it takes too long to process a request, thus increasing overall response time and decreasing throughput.

In conclusion, there may be scenarios where obtaining query results from a collection of partial subsets could be a benefit. However, care must be taken when implementing such an algorithm to prevent the Tracker restricting downloader choice or becoming a system bottleneck.

5.2 Conclusions

This paper builds on our previous introductions to the Wigan Peer-to-Peer database system and focuses specifically on the issue of how newly submitted queries are matched to existing adverts so that the downloading peers are only informed about those uploaders that can answer their query. We have discussed the format of the internal database maintained by the Tracker and how the Tracker uses this data to match queries to advertisements. Finally, we have suggested a possible extension to the Tracker which would consider any advert A in response to a query, Q if $A \subseteq Q$.

This research has highlighted the complexity of matching database queries in a P2P database compared with matching files in a file-sharing system. However, our initial results [1, 2, 3] have shown that this is an approach worth pursuing further.

References

1 J. Colquhoun and P. Watson. *A Peer-to-Peer Database Server.* In. A. Gray, K. Jeffery and J. Shao (eds). *BNCOD 25.* 2008, Springer.

2 J. Colquhoun and P. Watson. *A Peer-to-Peer Database Server based on BitTorrent.* In: *UK e-Science Centres All Hands Meeting, 2008, Edinburgh.*

3 J. Colquhoun and P. Watson. *Evaluating a Peer-to-Peer Database Server based on BitTorrent.* In. A. P. Sexton (eds). *BNCOD26*. 2009, Springer.
4 *BitTorrent.* Available at: http://www.bittorrent.com
5 R. Schollmeier. *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications.* In: *First International Conference on Peer-to-Peer Computing (P2P '01), 2001,* pp.101-102. IEEE Computer Society Press.
6 *Memcached.* Available at: http://www.danga.com/memcached/
7 D. Alvarez, A. Smukler and A. A. Vaisman. *Peer-To-Peer Databases for e-Science: a Biodiversity Case Study.* In: *20th Brazilian Symposium on Databases, 2005, Federal University of Uberlândia.* UFU.
8 R. Huebsch, J. M. Hellerstein, N. Lanham, et al. *Querying the Internet with PIER.* In: *VLDB '03, the 29th International Conference on Very Large Databases, 2003, Berlin.* pp.321-332. Morgan Kaufmann.
9 R. Huebsch, B. Chun, J. M. Hellerstein, et al. *The Architecture of PIER: an Internet-Scale Query Processor.* In: *CIDR 2005, Second Biennial Conference on Innovative Data Systems Research, 2005, Asilomar, California.*