



COMPUTING SCIENCE

A P2P Database Server Based on BitTorrent

John Colquhoun and Paul Watson

TECHNICAL REPORT SERIES

No. CS-TR-1183

January 2010

A P2P Database Server Based on BitTorrent

J Colquhoun, P Watson

Abstract

There are many instances when the dissemination of data sets to a wide audience may be beneficial. For example, in the scientific community this would serve as an opportunity to gain verification of results or allow experts to apply different evaluation techniques on the data generated by others. As computational devices are commonly employed in such analysis and evaluation, easing the electronic dissemination of such data is a necessity.

One method to achieve data dissemination is via a centralised database accessible over public access computer networks (e.g., Internet). In such a scenario data is stored on a centralised service which allows the retrieval of data as and when required by clients. However, in a centralised approach it may be possible for client requests to rise beyond the level that may be handled by a service. To prevent this occurring a scalable solution is required.

Commercial solutions are available that may scale to handle large numbers of client requests. Data is stored in a structured manner, allowing clients to form requests as queries to retrieve specified data items.

Unfortunately, any organisation hosting such a service must carry the financial cost of the additional hardware and expertise to maintain the scalable service. If data is popular and is required to be freely available (e.g., historical climate change data), this financial cost will be significant. Therefore, any solution that afforded similar scalability benefits while allowing query based retrieval of data for clients without the financial costs incurred would be desirable to many organisations.

Disseminating information in a scalable manner while limiting the financial burden on any one organisation is possible via peer-to-peer file sharing. Peer-to-peer file sharing software allows users to share files with each other without the need for such files to reside centrally. Files reside on the machines of users which cumulatively make up the peer-to-peer network of nodes. When a user request is made for a file those nodes that can satisfy the request may respond. The duplication of files across different nodes affords the scalability of data dissemination as potentially many nodes can service client requests. Cost of maintenance of the peer-to-peer network is, in essence, shared across all users.

In this paper we present a software system called Wigan. Wigan adapts the algorithms of the BitTorrent file-sharing protocol to achieve scalable database style information dissemination. In this approach we are removing the financial burden of maintaining a

scalable server. As such, Wigan will scale as, and when, more clients request data (join the network). Our approach differs significantly when compared to existing peer-to-peer file sharing techniques as we are dealing with data sets that are the result of client queries as opposed to file instances. This solution is challenging as Wigan must handle requests for data sets that may vary on a per-client basis compared to file instances which do not vary from node to node. In this paper we illustrate our approach to these challenges and present results showing that Wigan can, in certain circumstances, outperform traditional Client-Server database systems.

Bibliographical details

COLQUHOUN, J., WATSON, P.

A P2P Database Server Based on BitTorrent

[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-1183)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE

Computing Science. Technical Report Series. CS-TR-1183

Abstract

There are many instances when the dissemination of data sets to a wide audience may be beneficial. For example, in the scientific community this would serve as an opportunity to gain verification of results or allow experts to apply different evaluation techniques on the data generated by others. As computational devices are commonly employed in such analysis and evaluation, easing the electronic dissemination of such data is a necessity.

One method to achieve data dissemination is via a centralised database accessible over public access computer networks (e.g., Internet). In such a scenario data is stored on a centralised service which allows the retrieval of data as and when required by clients. However, in a centralised approach it may be possible for client requests to rise beyond the level that may be handled by a service. To prevent this occurring a scalable solution is required.

Commercial solutions are available that may scale to handle large numbers of client requests. Data is stored in a structured manner, allowing clients to form requests as queries to retrieve specified data items.

Unfortunately, any organisation hosting such a service must carry the financial cost of the additional hardware and expertise to maintain the scalable service. If data is popular and is required to be freely available (e.g., historical climate change data), this financial cost will be significant. Therefore, any solution that afforded similar scalability benefits while allowing query based retrieval of data for clients without the financial costs incurred would be desirable to many organisations.

Disseminating information in a scalable manner while limiting the financial burden on any one organisation is possible via peer-to-peer file sharing. Peer-to-peer file sharing software allows users to share files with each other without the need for such files to reside centrally. Files reside on the machines of users which cumulatively make up the peer-to-peer network of nodes. When a user request is made for a file those nodes that can satisfy the request may respond. The duplication of files across different nodes affords the scalability of data dissemination as potentially many nodes can service client requests. Cost of maintenance of the peer-to-peer network is, in essence, shared across all users.

In this paper we present a software system called Wigan. Wigan adapts the algorithms of the BitTorrent file-sharing protocol to achieve scalable database style information dissemination. In this approach we are removing the financial burden of maintaining a scalable server. As such, Wigan will scale as, and when, more clients request data (join the network). Our approach differs significantly when compared to existing peer-to-peer file sharing techniques as we are dealing with data sets that are the result of client queries as opposed to file instances. This solution is challenging as Wigan must handle requests for data sets that may vary on a per-client basis compared to file instances which do not vary from node to node. In this paper we illustrate our approach to these challenges and present results showing that Wigan can, in certain circumstances, outperform traditional Client-Server database systems.

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.

John arrived in the School of Computing Science as an undergraduate in 2001 following completion of the Year in Industry Scheme, during which he worked at Lite-On Ltd. On completion of his BSc in 2004, John began his

PhD, supervised by Paul Watson. His research project concerned distributed query processing in peer-to-peer database systems. He submitted his Thesis in October 2007 and from then until December 2008, worked as an RA, still in the field of P2P database systems. He was awarded his PhD in December 2008.

In January 2009, John 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

P2P COMPUTING
DATABASES

A P2P Database Server Based on BitTorrent

John Colquhoun
Newcastle University
School of Computing Science
Newcastle University
NE1 7RU
United Kingdom
John.Colquhoun@ncl.ac.uk

Paul Watson
Newcastle University
School of Computing Science
Newcastle University
NE1 7RU
United Kingdom
Paul.Watson@ncl.ac.uk

There are many instances when the dissemination of data sets to a wide audience may be beneficial. For example, in the scientific community this would serve as an opportunity to gain verification of results or allow experts to apply different evaluation techniques on the data generated by others. As computational devices are commonly employed in such analysis and evaluation, easing the electronic dissemination of such data is a necessity.

One method to achieve data dissemination is via a centralised database accessible over public access computer networks (e.g., Internet). In such a scenario data is stored on a centralised service which allows the retrieval of data as and when required by clients. However, in a centralised approach it may be possible for client requests to rise beyond the level that may be handled by a service. To prevent this occurring a scalable solution is required.

Commercial solutions are available that may scale to handle large numbers of client requests. Data is stored in a structured manner, allowing clients to form requests as queries to retrieve specified data items. Unfortunately, any organisation hosting such a service must carry the financial cost of the additional hardware and expertise to maintain the scalable service. If data is popular and is required to be freely available (e.g., historical climate change data), this financial cost will be significant. Therefore, any solution that afforded similar scalability benefits while allowing query based retrieval of data for clients without the financial costs incurred would be desirable to many organisations.

Disseminating information in a scalable manner while limiting the financial burden on any one organisation is possible via peer-to-peer file sharing. Peer-to-peer file sharing software allows users to share files with each other without the need for such files to reside centrally. Files reside on the machines of users which cumulatively make up the peer-to-peer network of nodes. When a user request is made for a file those nodes that can satisfy the request may respond. The duplication of files across different nodes affords the scalability of data dissemination as potentially many nodes can service client requests. Cost of maintenance of the peer-to-peer network is, in essence, shared across all users.

In this paper we present a software system called Wigan. Wigan adapts the algorithms of the BitTorrent file-sharing protocol to achieve scalable database style information dissemination. In this approach we are removing the financial burden of maintaining a scalable server. As such, Wigan will scale as, and when, more clients request data (join the network). Our approach differs significantly when compared to existing peer-to-peer file sharing techniques as we are dealing with data sets that are the result of client queries as opposed to file instances. This solution is challenging as Wigan must handle requests for data sets that may vary on a per-client basis compared to file instances which do not vary from node to node. In this paper we illustrate our approach to these challenges and present results showing that Wigan can, in certain circumstances, outperform traditional Client-Server database systems.

P2P Computing, Databases.

1. INTRODUCTION

The scalability of applications that place a heavy load on database servers has again become the subject of intense commercial and research interest. Systems that allow thousands of simultaneous users to browse and purchase goods require highly-scalable, multi-tier

systems, and so place great strain on the database tier. In another area, scientific researchers are now encouraged to provide open access to their databases so results can be widely shared, but this can cause performance problems if the data proves popular. The limitations of server scalability as the number of simultaneous accesses increases used to be a

problem in another area – file-sharing. However, that has been very successfully addressed in recent years by the introduction of Peer-to-Peer (P2P) techniques that harness the power of the clients in order to reduce the load on the server. This has led to the design of extremely scalable, reliable and widely used applications.

In this paper we describe Wigan – a P2P database system designed to investigate whether the techniques used by file-sharing systems such as BitTorrent [1] can be applied to building highly scalable access to databases. We believe that this work is timely as almost all client computers, including desktop PCs, now have significant quantities of spare resources (CPU, memory, disk, network bandwidth) that could potentially be used to reduce the load on a DBMS. In Wigan, clients cache the results of their queries and these are then used to answer subsequent queries from themselves and other clients, so reducing the load on the server. This is not limited to exact query matches (as in Memcached [2]) – peers can answer queries that are a subset of the results they have cached. Wigan is, to our knowledge, the first P2P database system designed with a focus on scaling up the performance of a single database server – whereas many existing P2P database systems focus on federating a collection of distributed databases. It is also, to our knowledge, the first P2P database which has been derived from the algorithms used in a file-sharing system.

Designing Wigan has proved challenging due to the major inherent differences between accessing files and querying databases. The main differences are:

- Database queries can include selects, projects and joins, whereas in file-sharing, files are accessed as a single unit,
- Query results may contain, within them, the results of other queries, whereas in file-sharing there is no equivalent,
- Databases are updated, whereas in file-sharing, files are considered immutable.

Clearly, there are therefore many issues that must be resolved in designing a P2P database. In this paper we explain how there are some complex issues that arise in P2P database processing that are not found in P2P file-sharing, which meant that we could not just produce our own implementation of the BitTorrent protocol. We then present solutions and evaluation results that show performance gains can be achieved.

This paper is structured as follows. Section 2 provides a brief overview of BitTorrent, Section 3 introduces the Wigan architecture, Section 4 discusses possible applications for Wigan and Section 5 presents experimental results. Section 6 introduces related work while Section 7 concludes this paper and provides suggestions for further work.

2. BITTORRENT OVERVIEW

BitTorrent is a hybrid P2P file-sharing protocol [3]. 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. THE WIGAN ARCHITECTURE

Wigan is based on 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 BitTorrent seed contains the complete copy of a file, whilst in Wigan a 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, 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. This ensures that all queries can be answered

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). When answering queries, the peers use an implementation of the BitTorrent choking protocol. 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. 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. Full details of the Tracker and its database can be found in [4].

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 [2]. 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
```

Both adverts can satisfy the query. We now describe in more detail the matching process. On arrival at the Tracker, the downloader's query is converted into the same canonical form as is used to store the adverts. The Tracker then retrieves all adverts which contain the tables and columns in the downloader's query. Note that if the downloader's query contains an aggregation, such as “MAX”, the Tracker will retrieve both advertisements with the same aggregation and those which have the original column values. This is because an uploader with either the original column values or the same aggregation will be able to resolve the query, the only difference being that the latter will not have to perform the aggregation again when the query arrives because it already has the result.

This initial selection process removes from consideration the advertisements which do not have all of the required columns or contain none of the tables that appear in the downloader's query. The Tracker then examines all of the advertisements it has retrieved in the initial selection process to check that the conditions in the “WHERE” clause of the advertisement do not prevent the advertisement from resolving the downloader's query. If the query and an advertisement each contain a join, the Tracker must also check that the advertisement contains all of the tables in the downloader's query, not just some of them. The result of the final part of the selection process is a collection of adverts which can all resolve the downloader's query.

This collection of adverts may include a selection of different queries that can satisfy the client's request. To enable a downloader to distinguish between adverts for different queries, the Tracker will group the adverts by query, stating for each query how many pieces the downloader should receive. This ensures the downloader is aware of when it has received all of the results.

3.4 Downloading and uploading

We now examine the process of downloading and uploading. The Tracker, using the processes described above, will have returned a list of suitable adverts grouped by query. 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. This is always the first group of peers, i.e. the group at the head of the list returned by the Tracker.

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. The issue of empty pieces will never arise in BitTorrent where the file-matching process only ever has two outcomes – either the files match or they do not. A file cannot partially match another file.

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 major difference is that 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 (i.e. the query is a proper subset of the advert), it will have to change the piece structure before it begins to upload and this process will now be described.

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 10 such students. There is no guarantee of how these students' details are distributed across the pieces; the downloader does not know in advance which pieces will contain data matching the query. However, when the downloader makes this data available to others, it would be inefficient to have the resulting 10 tuples still spread over 20 pieces. Instead, these 10 tuples could all be grouped together in the minimum number of pieces that will hold them.

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. APPLICATIONS FOR WIGAN

In this section, we discuss applications which may be suitable for our Wigan system. We also suggest scenarios where Wigan would not offer an improvement.

4.1 e-Science

One potential application that we believe may suit Wigan's characteristics concerns e-Science databases. A database with Internet access provides a good means of making research data accessible; however there are currently not a large number of large-scale e-Science databases, such as the SkyServer [5], available for researchers to query. The cost of buying and maintaining a large database server could be prohibitive. If the data proves popular, response times may become unacceptable (and therefore offputting to those who may submit queries) unless the researchers publishing the data invest in a more powerful server. Wigan could offer an alternative here. By allowing users to answer each others' queries, this will not only reduce the load on the original database server but may also prevent the need to upgrade it. e-Science data would also be very suitable to share via Wigan because it may not change very frequently (see Section 4.3). This may encourage more researchers to make research data available in this form.

4.2 e-Commerce

Another potential scenario concerns e-commerce applications which allow users to browse and purchase items online. These applications normally have a multi-tier architecture [6] to allow separate scaling of the application logic and the database servers. These database servers must be able to handle a very large number of simultaneous users; however recent experience has shown that the demands placed upon some database servers can become leading to poor performance [7]. Purchasing additional servers is a costly solution suffering from the limitations described above. Other solutions include database fragmentation [7] – known as “sharding” – or caching data in the main memory at the application tier. The problem with these solutions is that they need to be designed in an application-specific manner; hence any changes to the application can lead to changes in the way in which it interacts with the database. This could mean that the existing caching or fragmentation strategy no longer works. In addition, main memory caching suffers from the same problem as purchasing a more powerful server in that the increased capability may still be insufficient. Further holding a cache in the application layer means that additional cache management operations now have to take place there.

A P2P database in an e-commerce application would allow the middle-tier application servers to answer each other's queries. Unlike existing systems which

cache data at each application server for that server's own use only, the P2P database would permit the servers to query each other and hence data cached by one server may be utilised by another requiring the same information. This would reduce the load on the database server and so could help to prevent the database server becoming a bottleneck.

4.3 Unsuitable applications

The Wigan system relies on peers caching the results of the queries they receive and then using these results to answer others' queries. Therefore, one of the first requirements for Wigan to work successfully is that there should be some overlap in the queries. If all queries were entirely different to each other then only the seed would be able to answer queries. If there were a large volume of queries, the seed would become overloaded and this result is an occurrence of the same problem that Wigan was designed to overcome. Note that this does not mean queries have to match exactly, as explained in Section 3.2.

Another major challenge in P2P databases concerns data updates. Our current implementation of Wigan assumes a static database where data does not change. Although we have an outline algorithm which could permit updates (see Section 7), we still do not believe that Wigan would be suitable for data which changes very frequently, e.g. transaction processing applications.

5. EVALUATION

5.1 Initial implementation

Our first implementation of the Wigan architecture took the form of a simulator, built using the SimJava tool [8]. A variety of experiments using the simulator are shown in [4] which analyse the behaviour of Wigan in great detail. In particular, it concludes that Wigan can deliver scalability in certain cases (see Section 4.3 for a discussion of some unsuitable applications); however there can be problems at the start, when only the seed is available to provide data, i.e. there are no other peers that have received the results of a query, if there a large number of peers submitting query requests.

5.2 Native Implementation

Following our work with the simulator, we constructed a "native", (non-simulated) version of Wigan, written in Java and using OGSA-DAI [9]. A variety of scenarios have been evaluated, and this section will describe some of these experiments.

5.3 The TPC-H benchmark

We chose to evaluate the system using a standard benchmark. The Transaction Processing Council's TPC-H Benchmark [10] contains sample data from a

manufacturing company's database. There are eight tables ranging in size from 25 to over six million tuples. There is a query workload that consists of 22 queries which are designed to be executed consecutively in different tests. These queries have certain parameters which are randomly generated during each test. In addition, there are also two refresh functions which insert and delete data from the database although as Wigan is not presently able to accommodate updates the refresh functions were not used in these experiments. Many of the 22 queries contain some advanced SQL features including subqueries, outer joins, views and inner joins involving many of the tables – sometimes up to seven in one query. We kept the TPC-H benchmark at Scale Factor 1.

5.4 Executing TPC-H queries in Wigan

Queries in the TPC-H benchmark use a wide range of basic and advanced SQL features. The Wigan implementation has focused on basic select, project, join and aggregate queries and so evaluation has concentrated on the six TPC-H queries that fall into this category. In addition, two further queries involve an SQL "SELECT" over the results of a subquery. Therefore, these two queries were implemented by having downloading peers obtain the results of the subquery from Wigan and then running the outer query locally on the results. For example, the query in Figure 1

```

"SELECT o_year, SUM(CASE
WHEN nation = '[NATION]'
THEN volume ELSE 0 END) /
SUM(volume)AS mkt_share
FROM
(SELECT
EXTRACT(year FROM o_orderdate)AS
o_year,
l_extendedprice*(1-l_discount)AS
volume, n2.n_name AS nation
FROM
part,
supplier,
lineitem,
orders,
customer,
nation n1,
nation n2,
region
WHERE
p_partkey = l_partkey
AND s_suppkey = l_suppkey
AND l_orderkey = o_orderkey
AND o_custkey = c_custkey
AND c_nationkey = n1.n_nationkey
AND n1.n_regionkey = r_regionkey
AND r_name = '[REGION]'
AND s_nationkey = n2.n_nationkey

```

```

AND o_orderdate BETWEEN date '1995-
01-01' AND date '1996-12-31'
AND p_type = '[TYPE]'
) AS all_nations
GROUP BY
o_year
ORDER BY
o_year;"

```

Figure 1 – a TPC-H query involving an SQL “SELECT” over the results of a subquery

was evaluated by each machine executing the subquery in Wigan (i.e. through the peers) and then running

```

"SELECT o_year, SUM(CASE
WHEN nation = '[NATION]'
THEN volume ELSE 0 END) /
SUM(volume) AS mkt_share"

```

(with the appropriate “GROUP BY” and “ORDER BY” clauses) over the tuples that were returned.

Rather than just ignore the other TPC-H queries, the Tracker routed them to the seed for evaluation. There was one further problem – we were unable to process all of the 22 TPC-H queries due to limitations in MySQL and therefore these queries could not be executed in the experiments presented in this section. In total, there were 15 queries that we could process, eight of these, as described above, could be executed through the peers in Wigan, while the remainder were routed to the seed.

5.5 Evaluating TPC-H queries in Wigan

In this set of experiments, all 15 of the queries that could be processed were executed once in three different experimental scenarios. In the first, queries were executed in MySQL [11], without Wigan, in a standard Client-Server configuration. In the second, queries were executed via Wigan, however, as the seed was the only uploader available at the start of the download period, it would have to execute all queries initially. In the final scenario, the queries were also executed via Wigan, however, another peer had already executed all of the queries in advance. This meant that there was initially another uploader available to answer those eight queries which could be resolved by the peers in Wigan and could be described as a warm-cache scenario. This was to investigate how Wigan behaved once the initial problem of only the seed being able to provide data, described in Section 5.1, was overcome. In all cases, five peers (excluding the cache-warming peer in the third scenario) requested the queries, each peer starting at ten second intervals. The experiments were repeated five times and an average taken. The overall average response times are shown in Figure 2.

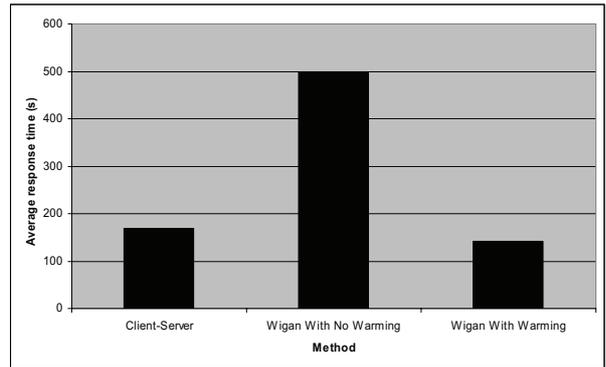


Figure 2 - Average response times for each method

Using Client-Server, the complex nature of some of the queries, involving multiple joins and subqueries does add to the overall average response time. However, again, routing everything to the seed in Wigan is considerably slower. This is due to those eight queries that can be executed by all of the peers. There are no uploaders apart from the seed, and for each piece request the seed receives, it must perform complex joins, in many cases over a large dataset. Given that, as described above, a table is composed of multiple pieces, the seed will have to perform the complex joins a number of times for each query, once for each piece. In Client-Server, this problem does not occur because the data is not being requested in pieces. However, when cache-warming is introduced to the Wigan system, the performance improves considerably. Indeed, by the time those peers starting later submit their queries; those starting earlier have received the query results and have advertised these through the Tracker, thus offering an even greater choice of uploaders. Having another uploader available, in addition to the seed, which can provide the query results initially, offers an improvement in performance for two reasons. Firstly, this uploading peer does not have to perform any complex joins because it already has the (joined) query results. Secondly, the query result set sizes are, in some cases, considerably smaller than the database tables and thus the uploader does not have to search through a large dataset or filter out any rows. Figure 3 decomposes these results and illustrates the average response times for those queries which can be answered by Wigan and those which can only be answered by the seed.

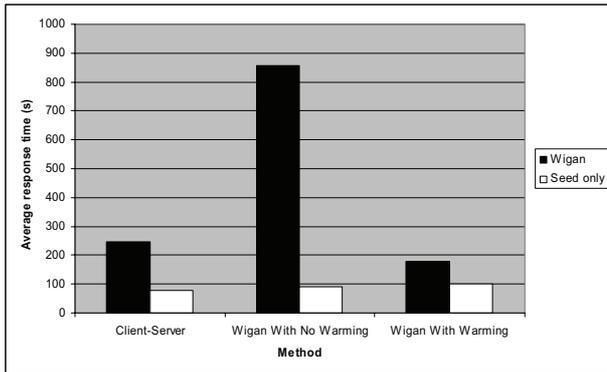


Figure 3 - Average response times for each method by query type

It can be seen that, for those eight queries that can be answered by the peers, the results follow the overall pattern shown in Figure 2. Client-Server is quite slow and using only the seed in Wigan is considerably slower again. However, the addition of a cache-warming peer offers a considerable improvement in performance as described above.

For those queries which can only be answered by the seed, Client-Server is slightly faster than Wigan, with or without cache-warming. For these queries, cache-warming will not make such a difference, because the queries cannot be answered by the cache-warming peer. Client-Server is faster because in Wigan, there is a slight overhead involved in contacting the Tracker and obtaining a list of peers, even though that list contains only one peer (the seed). This is not the case in Client-Server, where query execution can begin immediately.

6. BACKGROUND AND RELATED WORK

Our primary goal is to construct a low cost scalable database service capable of satisfying client queries. Utilising the processing capabilities of client machines could afford such an opportunity. This observation is based on the scalability achieved by Internet based peer-to-peer file sharing systems.

In this section we describe work related to our goal by starting with a description of peer-to-peer file sharing applications. We then describe existing database solutions that utilise Internet wide distribution in their architecture. Such solutions we categorise as caching, materialised views and Peer-to-peer. We conclude this section by summarising the required advancements required to realise a scalable database service using peer-to-peer file sharing techniques, emphasising the differences between our approach and existing distributed Internet database technologies.

6.1 P2P file-sharing

In recent years, there have been many P2P file-sharing systems which have been introduced, some more successfully than others.

Our work is derived from BitTorrent [1], which was discussed in Section 2 of this paper. Analyses of BitTorrent have concluded that it is a robust and scalable means of disseminating data, for example [12, 13]. However, the differences between sharing files and disseminating database data have meant that we have not been able to simply implement our own version of the BitTorrent protocol but instead we have had to make several changes as noted in Section 3.

Indeed these differences between file-sharing and sharing database data would also apply to other P2P file-sharing approaches. Some of these, such as Gnutella [14], a description of whose protocol can be found at [15], would pose even more of a problem to implement as there are no central components (such as the Tracker) at all and therefore each user would have to store their own records of which users have which query results.

6.2 Caching

Concepts to overcome the problem of server overloading have been examined within the area of traditional Client-Server databases. Various works have examined concepts related to caching, for example Dar et al [16] who propose a semantic model for client-side caching. The approach taken in [16] is to send only the “remainder query” to the backend database server, i.e. the part of the query which cannot be answered by the cache, and not the query as a whole. In this, it is similar to the Query Difference Operator [17], a concept designed to obtain the part of a query which cannot be answered by existing local resources. In Wigan, all queries should be answerable because the seed can answer all queries and the Tracker only suggests peers that can provide the required data.

The concept of dynamic caching has also been examined, for example IBM's DBCache [18] system which provides a local cache for some remote database. For all queries, two query plans are created – one using the local cache and one using the remote database. If the local cache does not contain the data, the latter plan is executed and the data is loaded into the cache for future use. DBCache is not designed for P2P database systems, but for Client-Server systems, as noted above. However, the ideas have some relation to the concept of Wigan, where the seed could be viewed as the remote, original, database and another peer which downloads data could be viewed as the local cache. If a downloader obtains data from the seed, this downloader is effectively receiving the data from the original database and making it available to other peers which may be geographically closer to it than the seed. However, there is a difference in the

way in which the cache is managed. The actual spread of data across the Wigan network would depend purely on the data which the downloaders download; in the same way that in BitTorrent, the extent that a file spreads across the network depends on the number of downloaders. In particular, there is no two-phased query plan produced in Wigan. All queries are routed directly to the Tracker which provides a list of possible options. It is up to the downloader to decide from which peer they actually obtain the data. In BitTorrent, the location of the possible peers is not taken into account at any time.

Microsoft's MTCache [19] is a similar approach, however a human administrator decides which data should be initially stored in the local cache. In addition, the cost of query evaluation is also examined in MTCache, so if for some reason it would be more efficient to execute the query remotely rather than locally, the local cache would not be used. This also bears resemblance to Wigan where different parts of a query may be received from different peers. However, like DBCache, this is a specific cache management system for Client-Server systems.

A third approach is bypass yield caching [20] which is specifically designed for scientific databases which could have a high workload. The aim of bypass yield caching is to reduce the overall network load on such databases. Local caches are placed near to the clients (i.e. the query submitters) and each cache acts independently. Ideas from the field of economics are used to determine whether to use a local cache or not, the cost being in terms of network traffic. If a request is made for data which is not in a local cache, a formula is used to determine, economically, if it is worth loading this data into the cache or not. BitTorrent and Wigan do not use economic formulae. Reducing overall network traffic is not a particular goal of Wigan.

An approach for implementing a dynamic cache in a P2P file-sharing system, the Distributed Cache Table is presented in [21]. Caches store digests of each document – containing key words about that document's contents – and interestingly, query subsumption is used. This means that the query results do not have to match the cache contents exactly, providing the query is a proper subset of the cache contents. However, any queries which cannot be answered by the caches are broadcast throughout the peers so that the results can be found. Initially all queries are broadcast until peers begin to populate their local cache. A mathematical formula is used to calculate the 'profit' of storing a query in the local cache, which takes into account factors including the size of the query results and the number of broadcasts made for that query. The latter statistic is an indicator of the query's popularity. The use of subsumption is similar to the way in which the Wigan Tracker matches queries to adverts as discussed in Section 3.3. However, the distributed cache table was designed for

a P2P file-sharing system and not a database. In addition, the distributed cache table assumes a system architecture with no equivalent of the Tracker. The Tracker, with its ability to match queries to adverts ensures there is no need for broadcasting in the Wigan system.

6.3 Materialised views

An alternative approach examined in Client-Server databases is to materialise views of the database. The concept of views as a whole is surveyed in detail in [22]. Chaves et al [23] consider the problem of materialised view selection for distributed databases. They divide databases using materialised views into three categories: those where the database and the views are on the same computer, those where the data may be on any computer but the views are all materialised on one machine and finally those where the data or views may be on any computer. If we view the seed in Wigan as being the original database and the data held by the peers to be the views, Wigan does not fit into any of these categories because the data is held on one computer but the views may be on any computer.

6.4 P2P Databases

The potential of P2P computing has attracted some interest in the database community. There are formal models and semantics of P2P databases, for example [24, 25, 26, 27], and also some existing P2P database systems, for example Piazza [28, 29], the Unified P2P Database Framework [30], PeerDB [31, 32] and Mapster [33]. There is also the PIER Internet query processor, [34, 35], which brings database query processing techniques into a non-database scenario. However, these all view a P2P database as a collection of distributed databases and focuses on federating these databases, for example through schema integration. This is different to Wigan which focuses on single database server scalability.

The BioWired P2P database system [36] focuses on federating a collection of databases owned by different organisations, though they must have a global schema. BioWired peers advertise their data and meet fellow peers – or 'acquaintances' – at rendezvous nodes. However, if none of a peer's acquaintances are able to solve a query, this query is unanswerable in the current BioWired system. In contrast, by using a Tracker, Wigan ensures that clients will always receive a result if that is possible.

The DÍGAME architecture [37, 38] allows peers to make their local databases available to other peers using a subscription service. Subscribing peers then have a replica of the database on their local machine and route any queries they have to this replica and not the original database. The database can only be updated by the originating peer which sends out a new

version of the database to the subscribers when an update occurs. A piece of middleware, known as the wrapper component, manages the schema integration between the original database and the replicas held by the subscribers. Although DÍGAME is similar to Wigan in that it allows peers to publish parts of their dataset, it is still designed as a means of combining data from autonomous databases and not a means of scaling a single database.

6.5 Summary

In this section, we have introduced many works relating to our goal of producing a low-cost and scalable database system.

Whilst there are many P2P file-sharing networks and protocols available, as discussed in this section and in Section 3 of this paper, these cannot be used as they are due to the number of differences between sharing files and sharing data; particularly due to the complex nature of database data when compared to files.

Within the domain of client-server databases, various solutions involving caching or materialised views have been developed to try and reduce the need to obtain data from the original database. However, we propose a P2P approach for scalability which should also be a low-cost solution and not require caches to be explicitly established on local machines and initially loaded with data.

There are some existing P2P database systems, however these are not designed to take a database and distribute it in a P2P manner. Instead, they concentrate on federating a collection of different databases, in many cases each with their own schemas, which are owned by separate organisations.

7. CONCLUSIONS AND FURTHER WORK

This paper has introduced the Wigan P2P Database System, a database architecture whose algorithms were derived from the popular BitTorrent file-sharing protocol. A central component known as a Tracker keeps a record of which peers have downloaded which queries and this information is used by query submitters to help them find peers which can resolve their queries. A special peer, known as the seed, possesses the complete database and can therefore answer any queries which are not held by the other peers.

The combination of the Tracker and the seed ensures that peers will always receive a correct and complete set of results to their queries. It is correct in that the results are the correct answer to the query and complete in that all tuples will be received.

Wigan is, to our knowledge, the first P2P database system designed for scaling up the performance of a

single database server, rather than on federating distributed databases. Wigan is also, to our knowledge, the first P2P database derived from the algorithms of a file-sharing system.

Our work designing and implementing Wigan has highlighted the differences between P2P file-sharing and P2P databases which we have had to accommodate and which we shall now describe.

- The query matching process which occurs at the Tracker is much more complicated in Wigan where the Tracker must ensure all adverts returned to a query submitter contain the correct tables, columns and tuples. However, in file-sharing, there are only two possibilities – the file requested by a user either matches exactly that being advertised by a downloader or it does not match at all.
- Queries may contain within them, the results of other queries. Therefore, in Wigan, a user may be receiving data from a peer which is advertising a different query.
- If this arises, then there may be some pieces returned to a downloader in Wigan which contain no tuples or which do not contain the maximum number of tuples that may fit on one piece.
- Therefore in Wigan, a peer may have to change the piece structure (in order to get the optimal number of pieces for that query) on receipt of the full query results. In BitTorrent this does not occur as the piece structure of a file does not change.
- In turn, this possible change of piece structure means that a peer cannot start to advertise a query until it has received all of the pieces. This is because it cannot predict in advance how many pieces it will be advertising.

From our research and evaluation, Wigan does not always outperform existing Client-Server database systems – as noted in Section 4.3, there are some unsuitable applications and also, as discussed in this paper and in [4], performance problems may occur when only the seed is available to provide data.

However, the ability to reduce the load on the seed and also the improved performance resulting from the ability to search through a smaller query result set instead of the whole table (when downloading data from a peer) can lead to a performance advantage over traditional database architectures.

There are many possible extensions to the existing Wigan system. Although the current Wigan Tracker only suggests adverts which either match or subsume the downloader's query, [4] suggests ways that the Tracker could include in its list of peers, those adverts which are partial subsets of the downloaders' query. We also plan to examine ways in which data updates

could be accommodated, also discussed in [4]. This is likely to involve a version numbering system and would permit different versions of a database (each dating from a different time period) to co-exist and is similar to MTCache, which permits users to set requirements as to how up-to-date they would like their data to be [39].

Overall, this work has shown that P2P database servers can outperform conventional DBMSs in particular cases. Our further work is focused on understanding the behaviour in more detail and using this to expand the range of cases for which a performance increase can be achieved.

8. ACKNOWLEDGEMENTS

We would like to thank our colleague, Graham Morgan for his advice and suggestions. The work described in this paper was funded in part by the European Union through the IST 2002-004265 Network of Excellence, CoreGRID.

9. REFERENCES

- 1 BitTorrent. <http://www.bittorrent.com>. (27/10/2009).
- 2 Memcached. <http://www.danga.com/memcached/>. (27/10/2009).
- 3 Schollmeier, R. (2001) A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. *First International Conference on Peer-to-Peer Computing (P2P '01)*, 101-102. IEEE Computer Society Press.
- 4 Colquhoun, J. (2009). A BitTorrent-Based Peer-to-Peer Database Server <http://www.cs.ncl.ac.uk/publications/trs/papers/1135.pdf>. (17/11/09).
- 5 SkyServer. <http://cas.sdss.org/dr6/en/>. (28.10.09).
- 6 Mohan, C. (2001) Caching Technologies for Web Applications. *VLDB 2001, 27th International Conference on Very Large Databases*, Rome, Morgan Kaufmann.
- 7 Henderson, C. (2006) *Building Scalable Web Sites*. O'Reilly, Sebastopol.
- 8 The SimJava tool. <http://www.dcs.ed.ac.uk/home/hase/simjava/>. (17/11/09).
- 9 OGSA-DAI. <http://www.ogsadai.org.uk/>.
- 10 Transaction Processing Council TPC-H Benchmark. <http://www.tpc.org/tpch/>.
- 11 MySQL. <http://www.mysql.com>.
- 12 Izal, M., Urvoy-Keller, G., Biersack, E. W., Felber, P., Hamra, A. A. and Garcés-Erice, L. (2004) Dissecting BitTorrent: Five Months in a Torrent's Lifetime. *PAM 2004: Passive and Active Network Measurement, 5th International Workshop*, Antibes Juan-les-Pins, April 19th-20th 2004, 1-11. Springer-Verlag.
- 13 Bharambe, A. R., Herley, C. and Padmanabhan, V. N. (2005). Analyzing and Improving BitTorrent Performance. <http://research.microsoft.com/~padmanab/papers/msr-tr-2005-03.pdf>.
- 14 Gnutella Protocol Development. <http://rfc-gnutella.sourceforge.net/>. (14/12/2009).
- 15 Description of Gnutella Protocol. <http://wiki.limewire.org/index.php?title=Gnutella>. (14/12/09).
- 16 Dar, S., Franklin, M. J., Jonsson, B. T., Srivastava, D. and Tan, M. (1996) Semantic Data Caching and Replacement. *VLDB 96, the 22nd International Conference on Very Large Data Bases*, Mumbai, September 3rd - September 6th 1996, 330-341. Morgan Kaufmann.
- 17 Minock, M., Rusinkiewicz, M. and Perry, B. (1999) The Identification of Missing Information Resources through the Query Difference Operator. *4th International Conference on Co-operating Information Systems (COOPIS 99)*, Edinburgh, 2nd - 4th September 1999, 304-314. IEEE Computer Society Press.
- 18 Altinel, M., Bornhövd, C., Krishnamurthy, S., Mohan, C., Pirahesh, H. and Reinwald, B. (2003) Cache Tables: Paving the Way for an Adaptive Database Cache. *VLDB 2003, 29th International Conference on Very Large Databases*, Berlin, 2003, 718-729. Morgan Kaufmann.
- 19 Larson, P.-A., Goldstein, J. and Zhou, J. (2004) MTCache: Transparent Mid-Tier Database Caching in SQL Server. *ICDE 2004 - 20th International Conference on Data Engineering*, Boston, 2004, 177-189. IEEE Computer Society.
- 20 Malik, T., Burns, R. and Chaudhary, A. (2005) Bypass Caching: Making Scientific Databases Good Network Citizens. *ICDE 2005, the 21st International Conference on Data Engineering*, Tokyo, 5th - 8th April 2005, 94-105. IEEE.
- 21 Skobeltsyn, G. and Aberer, K. (2006) Distributed cache table: efficient query-driven processing of multi-term queries in P2P networks. *International workshop on Information Retrieval in Peer-to-Peer Networks*, Arlington, 33-40. ACM Press.
- 22 Halevy, A. Y. (2001) Answering queries using views: A survey *VLDB Journal*, 10, 270-294.
- 23 Chaves, L. W. F., Buchmann, E., Hueske, F. and Böhm, K. (2009) Towards materialized view selection for distributed databases *12th International Conference on Extending Database Technology: Advances in Database Technology*, St Petersburg, 1088-1099. ACM.
- 24 Serafini, L., Giunchiglia, F., Mylopoulos, J. and Bernstein, P. A. (2003). Local Relational Model: A Logical Formalisation of Database Co-ordination. <http://eprints.biblio.unitn.it/archive/00000347/01/002.pdf>.
- 25 Franconi, E., Kuper, G., Lopatenko, A. and Serafini, L. (2003) A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems. *VLDB International Workshop on Databases*,

Information Systems and Peer-to-Peer Computing (DBISP2P'03), Berlin, 7th-8th September 2003, Springer-Verlag.

26 Majkic, Z. (2004) Weakly-coupled ontology integration of P2P database systems. *The First International Workshop on Peer-to-Peer Knowledge Management*, Boston, CEUR-WS.org.

27 Giunchiglia, F. and Zaihrayeu, I. (2002) Making Peer Databases Interact - A Vision for an Architecture Supporting Data Coordination. *CIA 2002, the 6th International Workshop on Cooperative Information Agents VI*, Madrid, 18-35. Springer-Verlag.

28 Gribble, S., Halevy, A., Ives, Z., Rodrig, M. and Suciu, D. (2001) What Can Databases do for Peer-to-Peer? *WEBDB'01, the 4th International Workshop on the Web and Databases*, Santa Barbara, 24th-25th May 2001,

29 Tatarinov, I., Ives, Z., Madhavan, J., Halevy, A., Suciu, D., Dalvi, N., Dong, X. L., Kadiyska, Y., Miklau, G. and Mork, P. (2003) The Piazza Peer Data Management Project. *ACM Sigmod Record*, 32, 47-52.

30 Hoschek, W. (2002) A Unified Peer-to-Peer Database Framework for Scalable Service and Resource Discovery. *Grid 2002, the 3rd International IEEE/ACM Workshop on Grid Computing* Baltimore, 18th November 2002, 126-144. Springer-Verlag.

31 Ng, W. S., Ooi, B. C., Tan, K.-L. and Zhou, A. (2003) PeerDB: A P2P-based System for Distributed Data Sharing. *19th International Conference on Data Engineering*, Bangalore, 633-644. IEEE Computer Society.

32 Ooi, B. C., Shu, Y. and Tan, K.-L. (2003) Relational data sharing in peer-based data management systems. *ACM SIGMOD*, 32, 59-64.

33 Rouse, C. and Berman, S. (2006) A Scalable

P2P Database System with Semi-Automated Schema Matching. *26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'06)* Lisbon, 78. IEEE Computer Society.

34 Huebsch, R., Hellerstein, J. M., Lanham, N., Loo, B. T., Shenker, S. and Stoica, I. (2003) Querying the Internet with PIER. *VLDB '03, the 29th International Conference on Very Large Databases*, Berlin, 9th-12th September 2003, 321-332. Morgan Kaufmann.

35 Huebsch, R., Chun, B., Hellerstein, J. M., Loo, B. T., Maniatis, P., Roscoe, T., Shenker, S., Stoica, I. and Yumerefendi, A. R. (2005) The Architecture of PIER: an Internet-Scale Query Processor. *CIDR 2005, Second Biennial Conference on Innovative Data Systems Research*, Asilomar, California, January 4th - January 7th 2005,

36 Alvarez, D., Smukler, A. and Vaisman, A. A. (2005) Peer-To-Peer Databases for e-Science: a Biodiversity Case Study. *20th Brazilian Symposium on Databases*, Federal University of Uberlândia, October 3 - 7 2005, UFU.

37 Laborda, C. P. d. and Popfinger, C. (2004) A Flexible Architecture for a Push-based P2P Database. *16th GI-Workshop on the Foundations of Databases*, Monheim, 01/06/04-04/06/04, 93-97. Universitat Dusseldorf.

38 Laborda, C. P. d., Popfinger, C. and Conrad, S. (2004) Digame: A Vision of an Active Multidatabase with Push-based Schema and Data Propagation. *GI-/GMDS-Workshop on Enterprise Application Integration (EAI'04)*, Oldenburg,

39 Guo, H., Larson, P.-A., Ramakrishnan, R. and Goldstein, J. (2004) Support for Relaxed Concurrency and Consistency Constraints in MTCache. *SIGMOD 2004*, Paris, 937-938. ACM.