A REVIEW OF DISTRIBUTED QUERY SYSTEMS IN CLOUD COMPUTING

Imran Ashraf
Department of Information Technology, University of The Punjab, Gujranwala Pakistan

ABSTRACT:

The size of the databases has grown beyond what can be managed efficiently by a single server. So, data distribution is the only viable option concerning performance and efficiency. Scalability is an important factors needed especially in distributed environment like grid or cloud computing. These paradigms expect almost infinite scalability which means at least thousands of sites. So we need systems which not only provide scalability but its automatic adaption as well, as scalability increases or decreases. This review paper presents the querying systems used to provide scalability, aggregation and fault tolerance.

Keywords: Cloud Computing, Distributed Systems, Distributed Query Processing, Scalability, Aggregation, Fault Tolerance, Network Communication, Information Retrieval

[1] INTRODUCTION

Many databases have outgrown what a single server system can provide and they continue to grow. To meet future demands for data storage we need to build database managements systems that can scale. However scaling is not easy. Distribution adds coordination overhead, and this overhead grows with the size of the system and limits scalability. A solution o the scalability problem is to make sites (i.e. nodes or computers) in the distributed system more autonomous and thereby reducing the need for coordination. The advent of grid and cloud computing promises easier access to computing resources than ever, with the possibility to scale the resource allocation up and down as needed [1,2]. The vision of distributed systems that automatically adapt resource usage to the user’s needs seems to be within reach but that requires distributed systems that can scale seamlessly from a single site to thousand of sites and adapt to the current workload [3, 4].

Distributed systems come with the option of varying data placement. The idea of a cloud system is that the he system scales with the load and storage requirement. This means that the workload and the number of sites available for data storage and query processing changes and the system should adapt automatically to these changes.

This paper is divided into three sections. After introduction section 2 gives an overview of distributed query system in general. Section 3 discusses in detail the querying systems selected for this paper. In the end conclusion is given.
[2] DISTRIBUTED QUERY PROCESSING

This section covers general distributed query processing methods. A more in depth description of such methods can be found in [5]. A query posed to a distributed system is assigned a coordinator site, e.g. the site that was contacted in order to pose the query. This could be the same site for all queries or the coordinator role could be distributed so that each query has its own coordinator. The coordinator processes the query by going through four basic steps. The coordinator starts by decomposing the query from it was posed by the user, typically SQL (Structured Query Language) to algebraic form that is used internally by the system. After decomposition, comes the data localization step where the coordinator site locates the data fragments necessary to process the query. The result is a basic query plan that explains where to get data and how to compute the final result. This query plan is improved by the query optimizer step. The optimizer chooses a strategy for executing the query including join order and how data fragments are scanned. Finally the optimized query is executed and a result is produced and returned to the user.

Query decomposition is a translation from one query language to the internal query language of the DBMS (Database Management System) and is mostly the same for both centralized and distributed systems, but the localization, optimization and processing steps deserve a closer examination. In a real system these steps are not necessarily separate but may integrate and overlap. However it is easiest to examine them one by one.

[2.1] DATA LOCALIZATION

Data localization is nonexistent in a centralized system since all data needed by the query are located on a single site. In a distributed system these data have to be found. The query describes which tables are used and in some cases also a range of values for an attribute in that table. Using this information, the localizer must find which sites store relevant data. Systems that store tuples directly in DHT (Distributed Hash Table) have rules for data placement and know that all sites potentially have relevant data. Data items are retrieved one by one and there is no need to look up which sites are involved in advance. However, if sites are allowed to store data independently of a DHT [6], a global catalog is needed.

The global catalog stores information on where data is located i.e. a mapping between tables or data ranges of tables and sites. This catalog may be centralized or distributed. In a large distribute DBMs an important part of the localization step is to rule out sites from the query. If sites are not ruled out each query has to be broadcast to all sites leading to inefficient use of the system. The localization step limits the involved site to those that are actually needed to process the query. The result of localization is a query plan that describes how and where to get and process data in order to get the final result

[2.2] QUERY OPTIMIZATION

Query optimization makes a series of improvements to the query plan. A significant optimization is to find a good join order since this often has a great impact on query execution time. Other optimizations involve access methods. The localization step has found relevant primary and secondary indices, and the optimizer may choose one of these or fall back on a complete scan of a table or a table fragment. These optimizations are not separate, so the existence of an index may affect join order. The optimizer also has the option of choosing replicas. It could choose the replica assumed to be the fastest to access, or it may chose to access multiple replicas in parallel to speed up access. A survey of query optimization can be found in [7].
[2.3] QUERY EXECUTION

There are two basic strategies for query execution in a distributed system: data shipping and query shipping. When using data shipping all query execution takes place on a single site. This may be the same site for all queries i.e. a central query execution site, but it can also be a different site for each query. In any case, there is only one site executing a single query. Other sites that are involved are used as data storage sites that only execute read and write operations.

The data shipping strategy moves data to the site of query execution. A query shipping system on the hand moves the query to the sites that store the data. If the result of one operator is used by an operator on another site, the site of the first operator will ship an intermediate result to the next site. In the end, the last operator produces the final result that is sent back to the user.

The optimizer decides which algebra operation are to be executed at which site. One strategy is to reduce the amount of data transferred between sites, which usually means that as many operation as possible are done locally where the data are stored. If data from two sites are used by an operator, the operator is located at the site that has the largest operand, resulting in only the smallest operand being sent over the network.

Query processing in large distributed systems present new problems to data owners. One is the issue of trusting a cloud provided site to execute queries correctly. Data owners can cryptographically sign data to make sure that changes are discovered but results of query operator are harder to verify [8, 9]. Also there is a question of privacy [10]. Not all data can be shared with the cloud provider and efficient execution of privacy sensitive queries is current research topic [11, 12].

[3] DISTRIBUTED QUERY SYSTEMS

The query systems provide query capability but not the permanent storage. Query processing done on data that is no permanently stored by the systems or produced on demand. The following section discusses the query system selected for this review paper.

[3.1] PIER

PIER the peer-to-peer information exchange and retrieval system [13, 14] is a general purpose query network for relational data. PIER uses a DHT as overlay network for storing tuples. These tuples will time out after a set time and sites that want to keep data in the PIER network has to republish their tuples regularly. The interface to the overlay network is algorithm agnostic and can be used with different DHT algorithms. PIER has been tested with several algorithms such as CAN [15], Chord [16] and Bamboo [17]. PIER objects are identified by an identifier consisting of a namespace, a key and a distinguishing suffix. The namespace is the name of the table or in some cases the name of a temporary result. The key is the table’s key attributed. These two parts are combined and used as input to the DHT’s hashing function. An additional suffix is used to make the identifiers unique in case two tuples hash to the same DHT key.

Tuples are stored as serialized Java objects and can use the full Java type system for storing data. Since tuples are not permanently stored in the network, they have to be refreshed regularly. A time to live value is associated with each tuple and sites have to renew tuples before they time out. If a site disappears from the network, its tuples eventually time out and are removed. To avoid this, it must hand its data over to another site before leaving the network.
A special dataflow language called UFL is used to specify queries as relational algebra graphs. The client is considered to be outside the PIER network and contacts a site in the network to pose its query. The site that is contacted is called the proxy site and acts as coordinator for the query. The UFL query is transformed into a set of Java objects representing the algebra operation. The coordinator site distributed this graph of Java objects to the others sites. When a site perceives a query on this form, it starts processing it and producing answer tuples. The tuples are continuously forwarded to the coordinator until the query stops after a time limit specified in the query.

Queries are distributed using a distribution tree. There exists one such tree in the PIER network, and this tree is used for all queries. Sites that connect to the network are added to this tree by sending a message to a hardcoded rot hash key. This message is routed through the DHT according to the routing algorithm, and the first site that receives the message adds the site as its child in the distribution tree.

**[3.2] ASTROLABE**

Astrolabe [18] is a robust information management system primarily designed for scalability management. However, it also supports powerful aggregation mechanism which enables users to build customized virtual databases based on peer-to-peer network [19]. It is a system for continuously running aggregation queries used for system monitoring. Astrolabe maintains aggregated data in a hierarchy of zones. Each zone stores and maintains aggregate values based on its sub-zones. At the lowest level, each site in the system constitutes a leaf. The network structure in astrolabe is unstructured. A site has a connection to its parent zone and automatically becomes a member of all ancestor zones. Each site also has connections to asset of other sites in its own zone and may have connections to other sites in the system. Each site constitutes a leaf zone. In this zone it has complete control over the data. In the higher tiers of hierarchy, zone stores and maintains aggregates over the data in sub-zones. The root of the hierarchy has aggregates that cover the whole system. A site querying the network will connect directly to the site that maintains the zone in question.

When a data item is updated in a leaf zone, an aggregate value in the leaf zone may change. Updates are distributed to other parts of the network using a gossip protocol that guarantees eventual consistency. To compute summaries of data, Astrolabe uses on-the-fly aggregation.
Using this protocol a site randomly selects another site with which it exchanges information about its closest ancestor zone. Each aggregate value is marked with a time stamp so that the gossip protocol knows which value is the newest. After this exchange aggregates of ancestor zones are computed if necessary. This gives eventual consistency but sites in the same zone may be inconsistent at any given time.

The gossip messages also include information about zone membership. This way sites learn about new sites that join the system. This information is crucial to keeping the network stable as sites come and go. Astrolabe processes queries formulated in SQL and provides ODBC and JDBC programming interfaces.

Figure: 2. A three level Astrolabe tree (root domains, child domain and leaf domains with attribute list) [20]

Working of Astrolabe looks like a database though it is a virtual database residing on a distributed network. Each domain in Astrolabe is just like a relational table which contains the attributes of its child domains. Astrolabe provides the abstraction of a single logical aggregation tree. It achieves its robustness by replicating all aggregated attributes values for a sub-tree to all nodes in that sub-tree. It may however lead to the problem that Astrolabe becomes unable to accommodate large number of attributes [21]. So, system becomes inflexible to support write-dominated attributes. Since astrolabe only provides eventual consistency guarantees, a query may return old data and subsequent queries may result inconsistent results. To ensure security and integrity of data Astrolabe makes use of digital signatures to point out corrupted data [20].

[3.3] SDIMS

The Scalable Distributed Information Management Systems (SDIMS) [21] is an aggregation system similar to Astrolabe, but based on DHT. Similar to Astrolabe all sites are leaves in a hierarchy of groups. Each site stores a selected set of attributes and aggregation functions to calculate the aggregates of these attributes for higher levels of the hierarchy.
SDIMS uses hierarchical aggregation for scalability instead of exposing info to all nodes. It aggregates information by use of reduction trees [18, 22] which allows nodes to access desired information and maintains scalability [19]. Instead of implementing a single static strategy like [15, 16, 22-24, 29] SDIMS provides flexible communication and propagation to support applications with variant requirements.

Where Astrolabe uses a gossip protocol to update sites SDIMS passes the aggregates up and down the group hierarchy. SDIMS API provides three functions namely install, update and probe. These operations take a parameter describing the requested propagation in the hierarchy, from local to global. This allows data items that are frequently read and infrequently written to be updated globally, so that probes (queries) can be done locally. On the other hand, items that are much more frequently written to can be updated locally, avoiding flooding the whole system when the values is most likely overwriten before the next time it is read.

SDIMS achieves robustness by using lazy re-aggregation and on demand re-aggregation. To ensure robustness it guarantees the principles of read availability and eventual consistency.

[3.4] TAG

The Tiny Aggregation Service (TAG) [22] is a generic aggregation service for wireless sensor networks. One site is appointed the root and broadcasts a message stating its own identifier and its level in the hierarchy. All sites that hear this and are not already in the hierarchy; adds the site as their parent and broadcast similar messages containing their own identifiers and levels which is one more than the parent’s level. As this process is repeated, the whole system forms a hierarchy. The messages are repeated periodically to detect new sites and to reconnect sites that have not been able to contact their parent for a while.

The hierarchy is used to do aggregation in two phases. In the first phase, the query is propagated downwards through the hierarchy. Each leaf site then computes its reply and sends it to the parent site. An intermediate level site gathers replies from its children, combines it with its own result and ships the result up on level to its own parent. In the end, the reply reached the root of the trees and is returned to the querying application.

The technique is not very tolerant to failures. If one site is unable to contact its parent, the whole sub-tree rooted at that site is lost from the reply. In [22] caching is proposed as a solution to increase the quality of the results. If a child has not replied within a given time
limit, its parent site will use the cached value. Since a site is allowed to choose a new parent if it is unable to connect to its current parent, care must be taken so that the cache is invalidated before the child can appear in another sub-tree.

TAG uses the basic facilities of databases query languages to provide a simple interface to elect and aggregate data. It is also capable of distributing and executing queries in a time efficient manner. Difference between SQL and TAG queries is that of output. TAG queries output is a stream of values instead of a single value. The chief advantage of using TAG is that it can decrease communication required to compute an aggregation.

[3.5] COUGAR

The Cougar Project [25-28] is another aggregation system for sensor networks and like TAG it also has a hierarchical structure where queries enter at the root site but the approach to query processing is quite different. At the top of the tree is the root site that issues all the queries. The internal nodes called view nodes of the hierarchy store pre-aggregated values for the next lower level, which are the leaf nodes with the actual sensors.

Queries are processed in a hybrid pull-push manner. View nodes proactively aggregate the values read from the leaves and queries are made against the view nodes. Two different types of connections are allowed: on-demand and proactive. In proactive links, information is pushed to update pre-aggregated values, while on-demand links require view nodes to pull the data from leaves. Querying is a three phase process. In phase one; leaves push data to the view nodes, which compute the aggregate values. In phase two, the query is sent out from the root to the view nodes. In the third and final phase, the view nodes send their replies to the root. The root then merges the partial results of all the view nodes to get the final result.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Features</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Features</td>
<td></td>
</tr>
</tbody>
</table>
| PIER | - General purpose query network  
- Intended for relational data  
- Based on DHT  
- Tested with algorithms like CAN, Chord, Bamboo etc |  
- Time outs require republishing  
- Tuples needs to be refreshed regularly  
- Site disappearance results in loss of tuples |
| ASTROLABE | - Unstructured structure  
- Routing protocol is Gossip [epidemic]  
- Ensures eventual consistency  
- SQL queries supported  
- Interfaces are ODBC(Oracle Database Connectivity), JDBC(Java Database Connectivity)  
- Replication supported  
- Use of digital signatures for security  
- API supports Find, Set and Get |  
- Sites in a zone may be inconsistent at any given time  
- Queries results may be old  
- Does not support atomic transactions |
| SDIMS | - Based on DHT  
- Passes aggregations up & down the group hierarchy | |

Imran Ashraf
**REVIEW OF DISTRIBUTED QUERY SYSTEMS IN CLOUD COMPUTING**

<table>
<thead>
<tr>
<th>TAG</th>
<th>COUGAR</th>
</tr>
</thead>
</table>
| - API functions: Install, Update and Probe  
  - Uses hierarchical aggregation for scalability  
  - For lower power distributed wireless networks  
  - Support SQL type queries  
  - Child cache  
  - Decreases communication required to computer aggregate  
  - Not failure tolerant  
  - In worse cases its performance is equal to centralized approach |
| - Hybrid pull-push method for view selection  
  - In network aggregation  
  - Low fault tolerance |

**[4] CONCLUSION**

Choosing optimal query system for the desired performance and efficiency is very crucial in cloud/grid computing. A review of some of the query systems is given along with a table describing their features and drawbacks with the hope that it will assist for the selection of an appropriate system.

**REFERENCES**


Author[s] brief Introduction

Mr. Imran Ashraf is a Lecturer at Information Technology Department of University of The Punjab, Pakistan. He got his MS in Computer Science from Blekinge Institute of Technology, Sweden in 2011. His research project was in Ericsson AB Sweden and was selected for “Sparbanksstiftelsen Kronan Scholarship” in 2011 which is awarded to most outstanding research project in Sweden every year. He also has a P.G.D in Information Technology in which he is a Gold Medalist.