Chapter 2

Background and Related Work

A traditional relational database stores data in tables which are then queried by users or applications looking for a subset of the data stored. A database management system (DBMS) provides additional functionality to help in organization, storage, retrieval and management of data. One fundamental property of such a database is that queries arriving on the database operate over a finite set of data. This ensures that the query results are returned to the query requester in a finite amount of time.

The advent of sensors and applications which generate data continuously however pose challenges which are significantly different that the ones offered by a traditional relational database. The key applications and drivers which generate unbounded data sequences as mentioned in Golab [33] are,

- The advent of sensor networks has become ubiquitous (Hellerstein et al. [40], Stonebraker et al. [82]) and are being used for monitoring geophys-
ical (Yao and Gehrke [94]) and environmental activities from volcanoes (Werner-Allen et al. [90]) to underwater acoustics (Pompili et al. [60]), road traffic monitoring (Arasu et al. [7], Madden and Franklin [52]), location tracking and surveillance (Abadi et al. [1], Wu et al. [92]) and inventory and supply chain management (Gonzalez et al. [34], Franklin et al. [28]). The use of sensors leads to continuous data streams being generated.

- The world wide web with its various data feeds for news, sports and financial tickers (Chen et al. [20], Lerner and Shasha [49]) coupled flexible aggregation mechanisms like Yahoo Pipes provide highly customized updates to users. The advent of newer application paradigms like Twitter generate substantial volume of streaming data as well.

- Continuous monitoring of transaction logs being generated by online transactions, point of sale devices, telephone calls and web-server logs (Gilbert et al. [32], Cortes et al. [22]) enable realtime detection of frauds.

- Analysis and monitoring of network traffic (Cranor et al. [23], Sullivan and Heybey [85]) helps in improving end user service quality and preventing denial of service attacks (Johnson et al. [42]).

The fundamental differences in traditional relational databases and stream data processing are summarized in Table 2.1.

The unique nature of streaming data processing leads to a number of challenges which are not present in relational databases and are discussed below.
<table>
<thead>
<tr>
<th>Property</th>
<th>Relational Databases</th>
<th>Stream Data Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of data</td>
<td>Finite</td>
<td>Unbounded</td>
</tr>
<tr>
<td>Frequency of data updates</td>
<td>On explicit modifications</td>
<td>Continuous</td>
</tr>
<tr>
<td>Ordering in data</td>
<td>None</td>
<td>Implicit (arrival time) or Explicit (generation time)</td>
</tr>
<tr>
<td>Computation of data</td>
<td>Pull Based (on query arrival)</td>
<td>Push Based (on query or data arrival)</td>
</tr>
<tr>
<td>Query Duration</td>
<td>Finite</td>
<td>Finite or Infinite</td>
</tr>
<tr>
<td>Query Plan Operators</td>
<td>Blocking</td>
<td>Non-Blocking</td>
</tr>
<tr>
<td>Duplicate Handling</td>
<td>Accurate</td>
<td>Approximate</td>
</tr>
<tr>
<td>System Conditions for a single query</td>
<td>Invariant</td>
<td>Can Vary</td>
</tr>
</tbody>
</table>

Table 2.1: Differences between Relational Databases and Stream Data Processing

1. The varying rate of data arrival at the stream query processing engine requires buffering and queuing mechanisms and is discussed in Motwani et al. [56], Madden and Franklin [52] and Abadi et al. [1].

2. Scheduling algorithms to handle the continuous nature of data (Babcock et al. [9], Carney et al. [17]).

3. Blocking operators cannot be used as they require to have the entire data set available before processing it Law et al. [47]. Examples of non blocking versions of query plan operators can be found in Avnur and Hellerstein [8], Madden and Franklin [52].

4. Adapt to changing conditions in the system by re-optimizing queries (Babu and Bizarro [11]) or by dropping a part of the incoming data during peak
loads (Babcock et al. [10], Reiss and Hellerstein [63], Tatbul et al. [87], Tatbul and Zdonik [86]).

5. Windowing of data to process queries and to archive old data is described in detail in Golab [33].

There have been attempts to use existing DBMS techniques for handling streaming data (Stonebraker et al. [82]). It is also possible to build an application layer over a DBMS to manage sliding windows, queries and data driven processing, however such an approach leads to inefficiencies leading to reduced scalability and increased query response times as shown in Arasu et al. [7].

Other approaches to handling streaming data discussed in Golab [33] using features available in DBMS use triggers (Hanson et al. [39], Paton and Diaz [58], Widom and Ceri [91]), arrays (Lerner and Shasha [48]), lists (Livezey and Muntz [50], Subramanian et al. [83]), sequences (Seshadri et al. [71, 72, 73]), time series (Schmidt et al. [66]) and temporal data (Zaniolo et al. [95]). While Babcock et al. [9], Arasu et al. [7] and Stonebraker et al. [82] discuss why triggers are not scalable and expressible enough for data streams, sequence, time series and temporal extensions assume that all the data is available in the database (Golab [33]).

Given the requirements for stream data processing, a data stream management system (DSMS) typically has an architecture as described in Figure 2.1. The buffer section incorporates buffering and queuing techniques to tackle the problem of varying data rates, the query plan section generates query plans and adapts the
plans generated to varying conditions. The execution section uses various non-blocking operators to execute the query plan generating the required results. The focus of this dissertation is in generating efficient query plans for LRC queries arriving on a set of grid nodes in an emergent fashion. While the work described so far provides the overview of stream data processing, the following sections describe previous work closely related to the work presented in this dissertation.

![Figure 2.1: A Generic Data Stream Management System Architecture(DSMS)](image)

### 2.1 Stream Query Processing

The TelegraphCQ project [19] builds on initial implementations of CACQ [53] and PSoup [18] to support an adaptive dataflow architecture using SteM [62] and Eddies [8] for continuous re-optimizations on a query. Eddies can also be shared across multiple queries. However, the eddies re-optimize based on the performance (tuple rates) of query modules (pipelines, non-blocking versions of standard relational operators). In contrast, the primary focus our work is on re-

24
ducing network usage. TelegraphCQ uses Flux [76] to load-balance and provide fault-tolerance. Data is exchanged between various modules using non-blocking queues provided by Fjords [52]. Borealis [2] is a distributed stream processing engine. The query processing is similar to the TelegraphCQ project. The operators in the query model can be distributed and optimization is done with respect to the placement of networks of operators that run continuously and interact with each other. The Borealis stream processing engine inherits its stream processing functionality from Aurora [16] and its distribution functionality from Medusa [96]. STREAM [6] is data stream management system (DSMS) where all data streams and queries arrive. STREAM processes queries by generating query plans for new queries and if possible merging it with existing query plans. Gigascope [23, 24, 25] is a DSMS from AT&T to monitor network traffic in real time.

The TinyDB [54], Cougar [93, 94] and related projects focus primarily on in-network aggregation and algorithms to reduce power consumption in sensors.

2.2 P2P Stream Data Processing

The StreamGlobe project [80, 46, 45] uses data sharing and in-network query processing to efficiently process queries in grids. By sharing stream data and reusing computational results, StreamGlobe reduces network traffic and computational load. Various cost models are described in Kuntschke and Kemper [44] using which relevant data sources are selected. The cost model in StreamGlobe
focuses on the additional network traffic and computational load caused by answering a query. The cost function used is of the form of,

$$C(P) = \alpha \cdot Bandwidth + (1 - \alpha) \cdot Load$$

where, \(C(P)\) represents the cost of query plan \(P\) and \(\alpha \in [0, 1]\) determines the dominant component of the cost function. A plan \(P_1\) is said to be better than another plan \(P_2\) if \(C(P_1) < C(P_2)\). There are however a number of issues if a static parameter like \(\alpha\) is used to determine the importance of bandwidth consumption and load. Considering the case where there are very few queries incident on the grid, the grid optimization objective should be bandwidth consumption only and not load distribution. Similarly when the grid is heavily loaded with a large number of queries, the optimization objective should be to only balance the load. These two scenarios indicate that \(\alpha\) cannot be a static parameter. Even if \(\alpha\) is made configurable and can change during the course of the query executions, it is not possible to continuously monitor the grid and modify the value of \(\alpha\) as to achieve desired overall optimization objective. This is because the exact relation between \(\alpha\) and its impact on the overall grid optimization objective is not known.

As the impact of local optimization on the overall grid optimization objective is difficult to model and predict, in our work we ensure that the grid as a whole is able to determine changing query patterns and adapt its local optimization strategies to ensure that the overall optimization objectives are met. Beside, StreamGlobe does not model autonomous behavior of the grid nodes. Once a
query plan is formulated by some node, other nodes that are contained as part of the plan will adopt the plan. In contrast, in our work, all nodes make autonomous decisions in self-interest, resulting in an emergent query execution plan. By allowing each node to make autonomous decisions, when a new query arrives or an existing query is revoked, only a small set of grid nodes are required to modify their query plans, independently, thereby reducing the impact of query arrivals and revocations on the entire grid. Finally, this work also takes into consideration de-queries and their effect in re-optimization of the data-stream routing.

In Seshadri et al. [74, 75] the authors consider the conflicting objectives of 1) minimizing communication and processing costs and 2) minimizing response times for query optimization in distributed data stream systems where multiple continuous queries execute simultaneously. The authors aim to reduce the search space for optimal placement of data stream operators by creating hierarchical network partitions. The authors also explore top-down and bottom-approaches to provide scalable query optimization. However, the cost function utilized by the work poses similar issues as in the StreamGlobe project and is discussed earlier.

### 2.3 DQP, CDNs and Multi-Query Optimization

Open Grid Service Architecture-Distributed Query Processing (OGSA-DQP) (Alpdemir et al. [4]) is a service based distributed query processing engine for grids. OGSA-DQP supports evaluation of queries using one or more database services provided on the grid. OGSA-DQP uses the now standard GDSs (Grid Discovery Services)
to get access to grid metadata and the databases on the grid. It uses techniques adapted from parallel databases to efficiently process queries (Smith et al. [79]).

In Shah et al. [77], repositories cooperate with each other and the sources to distribute dynamic data with coherence preservation, but do not consider de-queries.

Multi-query optimization in Sellis [69], Gupta et al. [37], Dalvi et al. [26], Roy et al. [64] and Gorman et al. [35] is done using scheduling, pipelining and caching techniques, which assume complete knowledge of the set of queries over which the optimization needs to be done and is primarily meant for centralized implementations.

2.4 Economic Paradigms

Mariposa (Stonebraker et al. [81]) is a wide-area distributed relational database. It uses economic paradigm for query processing and data migration and does not work with stream data. Economic models for resource allocation (Buyya et al. [13], Subramoniam. et al. [84]) primarily deal with the allocation strategies of grid resources to various clients but do not focus on stream data-sources.

2.5 Network Aware Query Processing

Network aware query processing techniques described in Ahmad et al. [3] and Pietzuch et al. [59] focus on the correct placement of operators in the network. Piet-
zuch et al. [59] introduces a spring relaxation technique to place operators in the network. It however does not consider replicated operators for distributing load, instead the load distribution algorithm provided tries to place the single operator on a suitable grid node. This would lead to overloading of the grid node if there a large number of queries on the grid requiring data from a single operator.

### 2.6 Complexity Analysis

Query optimization in databases is a very well studied area in data management. Original query optimizers search the plan space using dynamic programming (Selinger et al. [68]), applying a number of heuristics to reduce the number of options and ensuring tractable optimization. One of the key observations made in Selinger et al. [68] was the notion of pushing down projections and sometimes selections in the query tree to reduce the data transfer between operators.

The focus of subsequent research was primarily on optimizing joins (Kossmann and Stocker [43], Ganguly et al. [29], Ioannidis and Kang [41], Schneider and Dewitt [67]) and plan enumeration with other operators in Starburst [38] and Volcano [36]. Cost models for optimization using resource consumption as a metric were described in Mackert and Lohman [51]. The inability of resource consumption models to incorporate operator parallelism led to the development of response time models as described in Ganguly et al. [29].

A framework for determining the complexity of a general class of distributed query processing is discussed in Wang and Chen [89], while the NP hard nature of
optimal materialized view selection based on selection granularities is discussed in Park et al. [57].

The concept of data, query and hybrid shipping were introduced in SHORE [15] and has evolved as the operator placement problem in network aware query processing (Ahmad et al. [3], Pietzuch et al. [59]). The two step optimization process where query plans are generated in parts at compile time and runtime is discussed in Carey and Lu [14] and Ganguly et al. [30].

Multi-query optimization in Sellis [69], Roy et al. [64], Shim et al. [78] provides heuristics for computing the query plan, while Gupta et al. [37], Dalvi et al. [26], Gorman et al. [35] discuss issues with scheduling, pipelining and caching techniques in multi-query optimization. The NP hard nature of computing multi-query optimization in databases is discussed in Sellis and Ghosh [70].

Sensor networks (Madden et al. [54]) and in network query processing (Yao and Gehrke [93]) primarily focus on aggregate query optimization to maximize the energy efficiency of sensors. The complexity of multi-query optimization for aggregate queries in sensor network is evaluated in Trigoni et al. [88].

Network aware query processing techniques described in Ahmad et al. [3] and Pietzuch et al. [59] focus on the correct placement of operators in the network. Pietzuch et al. [59] introduces a spring relaxation technique to place operators in the network integrating the two step optimization process into a single step optimization process.