Task - G2[Team Report on a Unique Book Chapter]
Group - 12
Principles of Distributed Database Systems
CS4225
Current Issues: Stream Data and Cloud Computing
Members:
2010111 Nazmus Sakib Sarkar
2010139 Sazzad Hossain Mohammad
2009111 Naser Abir Chowdhury

Chapter-18
Current Issues: Stream Data Management and Cloud Computing
18.1 Data Stream Management

18.1 Data stream management

18.1.1 Stream Data Models

Data streams are ordered sequences of time-stamped tetuples. They can be modeled as relations, with a time-stamp attribute, or as collections of time-based groups. Various window models and data models handle unbounded streams and changing distributions.

18.1.2 Stream Query Languages

Persistent stream queries can be expressed in extended DBMSs. Monotonic queries can be incrementally updated, while non-monotonic queries require full recomputation. Querying paradigms include declarative, object-based, and procedural languages.

18.1.2.1 Declarative Languages

CQL supports diverse operators in the STREAM DSMS, while CSQL is used in Gigascope for selection, aggregation, join, and stream merge. SimaQuel in Telegraph CQ features a for-loop and windowing. Outputs are time-sequenced sets or streams.

18.1.2.2 Object-Oriented Languages

In Tribeca, streams are classified based on a type hierarchy; and the query language supports projection, selection, aggregation, multiplexing, demultiplexing, and joins. In COUGAR, sources are modeled as ADTs, and the query language includes signal processing and re-execution frequency.

18.1.2.3 Procedural Languages

In Aurora DSMS, users visually construct query plans using a graphical interface. SQuAl query language supports various operators, including filter, map, union, buffered sort, windowed aggregates, binary band join, and resampling.

18.1.2.4 Summary of DSMS Query Languages

The summary of DSMS query languages in Figure 18.2 highlights their inputs, outputs, operators, window types, and re-execution frequency. CQL offers diverse semantics, while SnapQuel supports extensive windows. GSQL, SQuAl, and Tribeca focus on streaming, and SQuAl and Aurora utilize ad-hoc processing.

18.1.3 Streaming Operators and their Implementation

Implementing streaming operators in DSMS involves non-blocking behavior, approximations, and sliding windows. Continuous query processing techniques help unblock operators. Sliding window operators handle arrivals and expirations, while join and aggregation operators update results accordingly.

18.1.4 Query Processing

The issues related to processing queries in Data Stream Management Systems (DSMSs) is discussed, where the overall process is similar to that of relational systems. Declarative queries are transformed into logical operators, which are then mapped logical operators specified in the query to physical implementations. It is presumed that both the inputs and operator state can fit in main memory.

18.1.4.1 Queuing and Scheduling

DSMS operators consume pushed data, while DBMS operators are pull-based. Queues facilitate data flow, allowing operators to consume tuples at their pace. Scheduling strategy assigns time slices to operators, ensuring they process tuples in timestamp order and deposit output tuples into the next operator's input queue. The scheduling strategy considers factors like tuple priority, processing cost, tuple size, and delay in reporting the answer relative to new data arrival.

18.1.4.2 Determining When Tuples Expire

Sliding window operators in DSMSs remove old tuples and update answers. Tuple expiration is determined based on the window's range. Joining tuples with different timestamps requires buffering. Weakening, Lifetime heights and expiration can vary for join results. The negation operator and count-based windows require explicit signaling of tuple expiration using negative tuples.

18.1.4.3 Continuous Query Processing over Sliding Windows

The negative tuple approach generates negative tuples for expirations, but it requires window materialization and doubles tuple processing. The direct approach avoids materialization and halves bandwidth, avoiding negative tuples, but may be slower for multiple windows due to sequential scans.

18.1.4.4 Periodic Query Processing Over Sliding Windows

Query processing over sliding windows stored in memory can be optimized by periodically advancing the window instead of continuously sliding it. The circular array model and techniques like the exponential histogram (EH) enable efficient computation of aggregates and operations. Although periodic evaluation may result in slightly stale results, it offers significant benefits. Overall, these approaches enhance query processing over sliding windows.

18.1.4.5 Query Processing over Windows Stored on Disks

In disk-based scenarios with periodically-sliding windows, optimizing index maintenance is crucial. Partitioning the data by insertion or expiration time localizes updates to specific disk pages. Chronological partitioning reduces index maintenance costs but scatters records across many pages, impacting query performance. Alternatively, a wave index divides the sliding window into equal partitions, separately indexing and clustering them by search key. This balanced approach improves data retrieval efficiency while maintaining reasonable index update times.

18.1.5 DSMS Query Optimization

The query optimizer in both DBMS and DSMS explores different execution strategies to choose an efficient one based on cost models and transformation rules. However, DSMS query optimization involves additional considerations such as adaptivity, load shedding, and resource sharing among parallel queries. The aim is to optimize performance and efficiency in processing queries within a streaming environment.

18.1.5.1 Cost Metrics and Statistics

Traditional DBMSs use selectivity and indices for efficient query plans based on disk accesses. In DSMS query optimization, cost metrics include response time, total time, output rate optimization, and quality-of-service metrics like response time. The goal is to optimize query execution in DSMSs to meet specific requirements and performance goals.

18.1.5.2 Query Rewriting and Adaptive Query Optimization

DSMS query languages introduce rewritings for operators. Adaptivity enables dynamic reordering based on system changes. Techniques like mid-query re-optimization and the Eddy approach enhance adaptivity and flexibility, accommodating changes. However, adaptivity comes with the trade-off of increased overhead for tuple routing and potential migration of internal state.

18.1.6 Load Shedding and Approximation

Load shedding in DSMS can be random or semantic, dropping less significant tuples. Approximation techniques like sampling and sketches can vary based on query plans. Optimal plans may change with load shedding. Tuples can be stored for later processing, and longer re-execution intervals act as load shedding.

18.1.7 Multi-Query Optimization

Memory usage in DSMS can be reduced by sharing internal data structures and computations. Shared operators execute a common plan, while multi-query optimization involves indexing predicates and matching tuples to satisfy multiple queries efficiently.

18.1.8 Stream Mining

Mining data streams poses challenges due to unbounded data sets, "messy" data, real-time processing, and evolving distributions. Techniques like sampling, load shedding, synopsis maintenance, and change detection are used. Tasks include clustering, classification, frequency counting, association rule mining, and time series analysis.

18.2 Cloud Data Management

18.2.1 Taxonomy of Clouds

Cloud computing provides on-demand resources and services over the Internet, categorized into Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). Enterprise-scale applications like OLAP are suitable for IaaS clouds, leveraging high-availability measures such as shared-nothing clusters and anonymization. Security and privacy are crucial in various cloud models, internal clouds, virtual private clouds, or hybrid clouds. Open source cloud software like Hadoop and Eucalyptus is emerging to promote interoperability and reduce vendor lock-in.

18.2.3 Cloud architectures

Cloud computing architectures vary based on the services provided (IaaS, PaaS, SaaS) and the business models of different cloud providers. Infrastructure components include infrastructure models relying on server virtualization, n-tier architectures, shared-disk clusters, and shared-nothing cluster architectures. Clouds offer functions such as security, directory management, resource management, billing, metering, pricing, accounting, and SLA management.

18.2.4 Data management in the cloud

Cloud data management recognizes that traditional relational DBMS technology may not be suitable for all cloud applications. Specialized DBMSs for cloud applications are emerging where they are needed. Internal clouds and virtual private clouds may still use traditional relational DBMS, but for OLAP and web-based applications, enterprises embrace prototyping scalability, elasticity, and high availability features. Key solutions include distributed file management, distributed database management, and parallel data processing. The cloud data management recognizes that traditional relational DBMS technology may not be suitable for all cloud applications. Specialized engines like column-oriented DBMSs and DSMSs are needed. Internal clouds and virtual private clouds may still use traditional relational DBMS, but for OLAP and web-based applications.

18.2.4.1 Distributed File Management

The Google File System (GFS) is a distributed file system developed by Google. It is designed for shared-nothing clusters, prioritizing fault tolerance, scalability, and large files and read/append operations. GFS organizes files as a tree of directories and uses fixed-size partitions called chunks. It employs a single master for metadata management and chunk replication for reliability. GFS supports read/append operations, ensuring data integrity through relaxed consistency model for concurrent writes and record appends. It ensures high availability through fast recovery and replication strategies.

18.2.4.2 Distributed Database Management

Online distributed database services like Amazon SimpleDB and Google Base allow users to manipulate structured data without defining a schema. Distributed database systems for cloud applications, such as Google Bigtable and Yahoo! PNUTS, prioritize scalability, high availability, and low latency over consistency and ease of development. Bigtable uses GFS for structured data storage, while PNUTS supports the relational data model and provides a replica consistency model. Both systems partition data horizontally and utilize replication for reliability.

18.2.4.3 Parallel Data Processing

MapReduce is a programming framework used for processing large datasets in a parallel and distributed manner. It simplifies parallel data processing by providing the map and reduce operations. MapReduce systems like Google's proprietary implementation and open-source versions like Hadoop leverage distributed file systems for storage. The flexibility of unstructured or semi-structured data, While MapReduce offers advantages in terms of scalability and fault-tolerance, its performance compared to parallel DBMSs has been a topic of debate.

18.2.2 Grid Computing

Grid computing provides web-based access to large compute and data resources. It emulates distributed and heterogeneous resources using web services. Grids are used in scientific and enterprise environments, with virtual administrators within Virtual Organizations (VOs). It enables collaboration, scalability, and cost-effectiveness. However, it is challenged by security and interoperability standards compared to cloud computing.

18.3 Conclusion

The topics of data stream management and cloud data management have potential but still require further research. Data stream management systems face challenges in scaling with data rates, distributed processing, stream data warehouses, and handling uncertain data streams. Cloud data management needs to address security, performance, interoperability, and ease of programming. Promising research directions include declarative programming languages, autonomic data management, data security and privacy, and green data management.