Distributed Multilevel Secure Data Stream Processing

Abstract— The data processing requirements of situation monitoring applications, which gather fast data from several sources, calculate findings instantly, and take action in real time, have given rise to the proposal of Data Stream Management Systems (DSMSs). The processing of military applications, which require data with numerous security levels and information flow regulations, is not appropriate for traditional DSMSs. Furthermore, military applications might not be a good fit for a centralised DSMS. We suggest a distributed multilevel secure DSMS system in order to achieve this. We provide an architecture, an implementation, and show how our system can process basic queries.

I. INTRODUCTION

Situation monitoring apps are becoming more and more commonplace because to the development of smart technologies and the widespread availability of sensors and mobile devices. For these kinds of applications, rapid data collection, processing, real-time result computation, and action are necessary. Data Stream Management Systems (DSMSs) have been proposed for such applications which enable the processing of streaming data and the execution of continuous queries. This system might be applied in military settings where data is collected by DSMS from several sources, including sensors and devices with varying degrees of security. We start this effort by outlining our approach to running continuous MLS queries. We provide a rough description of a few sample queries in our system. Next, we present our distributed processing architecture and the techniques that we employed to share the burden among many servers. We provide our prototype's information. Our solution enables the efficient and safe processing of MLS continuous inquiries.

We must address a number of research difficulties in order to design an MLS DSMS. A continuous query language is required to represent MLS DSMS inquiries in the real world. Keep in mind that users with varying degrees of protection will receive various answers to the same inquiry. Furthermore, query processing needs to be effective in order to satisfy a DSMS's QoS requirements. Denial-of-service attacks can be avoided and performance can be increased with the use of distributed query processing. Hence, a total redesign or significant alterations to DSMS components are required to process MLS continuous inquiries in a secure and effective manner.

This is how the remainder of the paper is structured. Related work is discussed in Section Il. In Section Ill, and an MLS formalisation model for stream processing is defined, along with the method of assigning security levels to queries, streams, and users. Additionally, it provides some sample questions and explains why these queries cannot be processed by current DSMSs. The adaptation of the centralised DSMS architecture for the processing of distributed MLS queries is demonstrated in Section IV. Details of the implementation are given in Section V. The paper is concluded in Section VI with recommendations for further reading.

Il. RELATED WORK

The works from closely related fields—DSMS, DSMS security, and MLS in real-time systems—will be covered in this part. Systems for managing data streams (DSMSs): The majority of the work done on DSMSs addresses a variety of issues, from theoretical findings to the implementation of thorough prototypes on how to manage data streams and provide responses in almost real-time without compromising service quality. Many studies have been conducted on the development of QoS delivery methods, including load shedding approaches as well as scheduling strategies. Borealis MayStream, and Stanford STREAM Data Manager Aurora are a few of the study prototypes. DSMS Security: Role-based access control has been used in a number of recent publications to secure DSMSs. These technologies do not stop illicit information transfers, even when they facilitate safe processing. Furthermore, unlike access control support, we must categorise every DSMS component in MLS systems. In, it is suggested that RBAC be enforced across data streams using punctuation. Every time, one or more security punctuations are used to convey access control regulations before to the actual data tuple. The permissions for a CQ are specified by query punctuations. A unique filter operator (stream shield) that is a component of the query plan handles both punctuations.

Research on creating a DBMS for the MLS real-time database system focuses on creating transactions with deadlines that run in serialisation order without causing data conflicts or security lapses. Our MLS DSMS has similarities with concerns like job scheduling and security breaches. Because real-time database management systems share data among transactions at multiple levels, covert channel difficulties must be addressed. In order to handle high-level transactions that clash with low-level transactions, several concurrent control protocols, such as OPT-Sacrifice, OPT-WAIT, and 2PL high priority, suspend or resume the high-level transaction. But if there are too many conflicts in the system, the hunger on high level transactions gets severe.

While DSMS manages continuous queries, real-time DBMS employs transient transactions for the system's execution unit. Transactions may establish a hidden channel or inference by accessing the same data item while persistent queries attempt to manipulate the response time, so causing a security breach. Whereas scheduling in CQ must take into account security, query response time, and throughput, scheduling strategy in MLS real-time transaction processing must take into account secu rity, serialisation, and transaction deadlines.

Ill. SECURITY MODEL

First, they introduced a multilayer secure (MLS) DSMS system paradigm. A partial order security structure, (L, \), is linked to an MLS DSMS. L represents a group of security tiers, while < denotes the dominance relationship among them. When Ll is less than 142, 102 is considered to strictly dominate Ll, and 142 is considered to strictly dominate Ll. The two levels are considered equal if Ll = 122. Ll 122 indicates Ll < 142 or Ll = 142. If Ll 142, then Ll is considered to be controlled by L2, and 142 is said to dominate Ll. If neither Ll L2 nor 142 Ll, then two levels Ll and L2 are considered incomparable. Users of the system are authorised for varying degrees of security. They indicate that user Ui has received L(Ui) security clearance. Take into consideration a configuration with two security levels: Low (L) and High (H),

where L < H. Jane Doe, the user, has a High security clearance. Specifically, L(JaneDoe) = H. There are one or more linked principles for every user. The user's security clearance determines how many principals they are affiliated with; this number is equal to the number of levels that the user's security clearance dominates. Jane Doe has two principals in our example: JaneDoe.H and JaneDoe.L. The user logs in as one of the principals for each session. The security level of the relevant principal is applied to any processes that the user starts during that session.

Generally speaking, three granularities—attribute, tuple, or stream—can provide multilayer security. Despite being the simplest method of enabling multilevel security, stream level enforcement—that is, single level streams inside the DSMS—does not function for a large number of MLS applications. We have examined stream applications across several domains (e.g., infrastructure security, battlefield monitoring). In these kinds of applications, streams of tuples at various levels are frequently fed into the DSMS. Thus, such applications would not benefit from stream level security. In this study, we apply security enforcement at the tuple level, meaning that we give each tuple a level. Therefore, in this work, we do not take into account each attribute's security degree separately.

IV. MULTILEVEL STREAM PROCESSING ARCHITECTURE

A. General DSMS Architecture:

A typical DSMS design based on the STREAM system. Query plans or specification languages can be used to create a Continuous Query (CQ). The input processor processes the CQs described using specification languages and produces a query plan. A directed graph of operators, such as Select, Project, Join, and Aggregate, makes up each query plan. An output queue and one or more input queues are connected to each operator. Each operator (such as Join) that must preserve the tuples' present state in order to evaluate the operator in the future is linked to one or more synopses. After that, the created query plans are instantiated and the query operators are set to the ready state, enabling their execution. The scheduler selects a query, an operator, or a path and initiates execution based on a scheduling strategy (e.g., round robin). When necessary, the run-time optimizer monitors the system and starts load shedding. These two QoS delivery strategies optimise throughput and performance while utilising less resources (such as queue size). In the DSMS, each stream has a stream shepherd operator that manages all of the tuples that enter that stream.

B. MLS Distributed Architecture:

We suppose that there are a number of servers in a distributed environment, each with a distinct id. Preassigned security levels are set for each server and are never altered. A group is made up of all the servers with the same degree of protection. A list of authorised users for that level is kept up to date by each server. As a result, a client at that specific level can send his query to any server. The master is the server that receives a query from a client. Slaves are the other servers on the same level. The query's execution is coordinated by the master. It sends the user the results that it has received from the slaves. Group members converse with one another on their status and workload. As a result, the master is able to balance the load for the specified queries.

V. PROTOTYPE IMPLEMENTATION

A. Server Communication:

A distinct server ID (Sid), IP address, port number, and pre-set Input Stream source security level Ls are all issued to each initial DSMS server. For communication, online servers with the same security level band together. A specific multicast group receives server information, such as sid and current CPU use (cu), on a continual basis. In order to obtain information about their peers, the servers listen to other members of the same group.

Only servers that belong to the same group may communicate with each other. Every server maintains an ip_table with the sid and cu of the group's available members. The ip_table is updated in real-time in accordance with changes in server availability and CPU utilisation.

B. Distributed Processing:

Every server keeps track of authorised users whose login level matches that of the server. When a user connects to a server by running client software, authentication is carried out. The only person who may perform additional actions is the one whose login and password are listed in the profile. Remember that the master server is the one to which a user connects successfully. Slaves are other servers that belong to the same group as the master. Slave servers get the query load from the master, which serves as a dispatcher.

Query registration: Once a user has been authenticated, the master uses all the available members in the group to act as slaves and provide processing power for queries submitted by user. The client begins to register input stream schemes in master and all slave servers. Besides, a query registration message from client is redirected to the interpretation unitin slaves which translates the interpreted query to a logical query plan (a link of operators).

Plan Generation: The master asks all slaves to optimise the crude physical query plans made in the preceding stage as soon as it gets a command from the client stating that all queries have been registered. Physical plan graphs are also created for user viewing. Every slave server's execution unit instantiates the produced physical plans.

VI. CONCLUTION:

Applications for situation monitoring need to gather and interpret fast data created in real time so that prompt action may be performed. To meet these applications' requirements for data processing, DSMSs were created. Secure military applications that handle data from several security levels and demand that there be no unlawful information flow between security levels cannot be processed by the DSMSs now in use. Our distributed distributed system management system (DSMS) design has the capability to handle data from several security tiers while preventing unauthorised information transfer. A load balancing approach is used to process user-submitted queries on servers that are on the same level.

We've created a basic prototype that shows how easy non-blocking queries can be used.

VII. DISCUSSION AND ANALYSIS:

The research paper addresses the evolving landscape of situation monitoring applications, emphasizing the need for efficient processing of high-speed streaming data in real-time. Traditional DSMSs are not well-suited for military applications, especially when dealing with data from multiple security levels and the necessity to adhere to strict information flow rules. The authors propose a distributed multilevel secure DSMS system to overcome these challenges.

The paper advocates for a shift in focus from traditional DSMSs that primarily address access control to more comprehensive MLS DSMSs that prevent illegal information flow across security levels. The authors contend that their proposed system not only prevents unauthorized access but also ensures the absence of covert and overt channels, providing a robust solution for military applications.

Innovation and Contribution: The proposed distributed multilevel secure DSMS system represents a novel approach to address the unique challenges posed by military applications. The emphasis on preventing illegal information flow and providing a secure and efficient solution distinguishes this work from traditional DSMS research.

Research Gaps and Challenges: The paper acknowledges several research challenges, such as the need for a continuous query language, efficient query processing, and distributed processing. However, it would be beneficial to delve deeper into potential limitations and areas for future improvement in the proposed system.

Comparison with Existing Solutions: While the paper discusses related work in the context of DSMSs and MLS in real-time systems, a more explicit comparison with existing solutions would enhance the paper's contribution. Understanding how the proposed system stands out from or builds upon previous research would strengthen the argument.

Real-world Applicability: The authors emphasize the relevance of their work in military applications. However, further discussion on how the proposed system could be applied in real-world scenarios, its potential impact, and practical deployment challenges would enhance the paper's practical significance.

VIII. REFERENCES:

1.B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in Proc. of the PODS, June 2002, pp. 1-16.

2.A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "Stream: The stanford data stream management system," Stanford InfoLab, Technical Report 2004-20, 2004. [Online]. Available: http://ilpubs.stanford.edu:8090/641

3.D. J. Abadi, Y. Ahmad, M. Balazinska, U. Ç etintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A.Rasin, E.Ryvkina, N.Tatbul, Y.Xing, and S. B. Zdonik, "The design of the borealis stream processing engine," in Proc. of the CIDR, 2005, pp. 277-289.

4.S. Chakravarthy and Q. Jiang, Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing, ser. Advances in Database Systems, Vol. 36. Springer, 2009.

5.M. Bishop, Computer Security: Art and Science. Addison-Wesley, December 2002. [Online]. Available: http://nob.cs.ucdavis.edu/book/bookaands