LOAD SHEDDING IN DATA STREAM MANAGEMENT SYSTEMS

In this chapter, we focus on a fundamental problem that is central to a DSMS. Namely, we investigate the problem of load shedding during temporary overload periods. This problem is not an issue in a traditional DBMS since:

- its queries are one-time queries;
- its data sources, although large, are static in nature;
- there is no support for QoS requirements for query processing.

In contrast, multiple CQs in a DSMS are active simultaneously for a long period of time. Theoretically, a CQ can be active in the system forever. The input rates of input streams are uncontrollable and highly dynamic in many stream-based applications. This highly dynamic input rate can prevent the system from keeping up with the tuple processing rate during high input rate periods. As a result, a large amount of unprocessed or partially processed tuples can be backlogged in the system, and tuple latency can increase without bound. Due to the predefined QoS requirements of a CQ, the query results that violate their QoS requirements may become useless or even cause major problems. Therefore, we need to limit the number of tuples buffered in the system so that all final query results satisfy their predefined QoS requirements. A feasible and viable solution to limit the number of buffered tuples in the system is to gracefully drop a portion of its unprocessed or partially processed tuples during high input periods, thereby relieving system load, and making it feasible to satisfy predefined QoS requirements. This process of gracefully dropping tuples is called load shedding. It should be pointed out that the accuracy of final query results is degraded due to loss of tuples in a load shedding process. Fortunately, many stream-based applications can tolerate some inaccuracy (i.e., approximate instead of precise results) in final query results if we are able to guarantee an upper bound on the inaccuracy. Generally speaking, load shedding is the mechanism for preventing final query results from violating predefined QoS requirements by discarding a portion of unprocessed or partially processed tuples. The predefined QoS requirements considered in this chapter mainly include the Maximal Tolerable Tuple Latency (MTTL)
and the Maximal Tolerable Relative Error (MTRE) of a CQ in its final query results.

In this chapter, we propose a framework and a set of techniques [126, 128, 132] for a general load shedding strategy by dynamically activating load shedders in query plans or deactivating active load shedders based on the estimation of current system load. These load shedders drop tuples in either a randomized manner or using user-specified application semantics. Specifically,

1. We exploit the optimal physical implementation of shedders in DSMSs with a goal of minimizing computation overhead and memory-consumption of a shedder.
2. We develop techniques to estimate system load to implicitly determine when load shedding is needed and how much needs to be shed.
3. We develop algorithms to compute the optimal placement of a load shedder in a query plan.
4. We also develop algorithms that determine how to distribute the total number of tuples (in terms of percentage) to be dropped among all load shedders with the goal of minimizing the total relative errors in the final query results due to load shedding.
5. Finally, we perform extensive experiments to validate the effectiveness and the efficiency of proposed load shedding techniques.

The rest of the chapter is organized as follows. Section 7.1 provides a formal definition of the load shedding problem. Section 7.2 discusses the detailed physical implementation of shedders. Section 7.3 describes our load estimation and load shedding algorithms. Section 7.4 presents a prototype implementation and the experimental results. Section 7.5 summarizes the chapter.

7.1 The Load Shedding Problem

In a multiple query processing system over streaming data, various scheduling strategies have been proposed either to minimize the maximum memory requirements [111, 131] and tuple latency [131] or to maximize the throughput [167]. A scheduling strategy can improve the usage of the limited resources in such a system. However, it can never improve the maximal computational capacity of a system, which is inherently determined by its fixed amount of resources such as CPU cycles, size of RAM, and so on. When the total load of active queries in a system exceeds the maximal computational capacity of the system, the query processing system has to either temporarily backlog some unprocessed or partial processed tuples that cannot be processed immediately in queues (or buffers) or discard them immediately. However, if the tuples are temporarily backlogged in queues, this causes a longer tuple latency, which is theoretically unbounded (as the queue size can grow indefinitely). Depending upon applications, this longer tuple latency may be unacceptable. Fortunately,