In this chapter, we focus on run-time resource allocation problem of stream data processing. Scheduling, a mechanism for the allocation of CPU cycles, determines the order in which operators or operator paths are scheduled at each time slot in a multiple-CQ processing system. The irregular and bursty input characteristics of data streams and the near real-time response requirements from stream-based applications further entail DSMSs to carefully allocate the limited resources in the system. Improper resource allocation can cause DSMSs to fail in handling temporary bursty data streams and in providing timely responses. As we will show in this chapter, improper resource allocation can cause delayed responses which may not be acceptable to many applications. Improper allocation of resources can also cause steep increase in maximal memory needed by the system and may even exceed physical memory available in the system resulting in system failures. However, these failures can be avoided with proper resource allocation mechanisms in place and proper capacity planning (as discussed in Chapter 5). The long-running characteristic of CQs and the unbounded nature of inputs further makes the resource allocation a run-time problem as decisions may have to be changed during the course of execution of a CQ. In short, it is clear that a resource allocation mechanism is critical to the success of a DSMS.

The scheduling problem in a DSMS is also a complicated one. First, it has significant impact on the performance metrics of the system, such as tuple latency, maximal memory requirement, and system throughput. We will define these performance metrics in Section 6.1.2 below. Second, although the resources such as memory size and CPU speed are fixed, scheduling can be highly dynamic (changing over a period of time) based on the observed performance metric values. Third, various predefined QoS requirements for a query add additional constraints to an already complex problem. Finally, the problem is a complicated one because the problem of finding the schedule that only minimizes the memory required is NP-complete as shown in [85]. Additionally, a desirable scheduling strategy in a DSMS should be able to: (i) achieve the maximal performance using bounded amount of resources;
be aware of the unexpected overload situations, and take corresponding actions in a timely manner; (iii) guarantee user- or application-specified QoS requirements for a query, if any; and (iv) be implemented easily, and run efficiently with a low overhead.

A single scheduling strategy is not likely to satisfy all of the above-mentioned performance metrics, as there are trade-offs among these performance metrics and the usage of limited resources. Although a large number of scheduling strategies exist in the literature for a broad range of needs, specific strategies have also been proposed for minimizing the maximal memory requirements in a DSMS. For stream applications, tuple latency is another important measure that is used in Quality of Service (QoS) specifications. In this chapter, we develop the following scheduling strategies:

1. the Path Capacity (PC) strategy to achieve the best overall tuple latency;
2. the Segment strategy to achieve lower maximal memory requirement than the PC strategy and better overall tuple latency than all operator-based strategies, such as an operator-level Round Robin strategy, the Chain strategy [85], and others;
3. the Memory Optimal Segment (MOS) strategy, which achieves the optimal memory requirement and further improves the maximal memory requirement of the Chain strategy, which has a near-optimal memory requirement;
4. the Simplified Segment (SS) strategy, which requires slightly more memory but much smaller tuple latency than the segment strategy;
5. the Threshold strategy, which is a hybrid of PC and MOS strategies.

The above strategies provide a reasonable overall performance although they do not meet all the desirable properties.

The rest of the chapter is organized as follows. Section 6.1 provides detailed discussion of our scheduling model, and a summary of notations used in this chapter and the rest of the book. Section 6.2 introduces some preliminary scheduling strategies and shows the impact of a scheduling strategy on performance (i.e., maximal memory requirement, tuple latency, throughput, and so on) of a DSMS. In Section 6.3, we first propose the PC strategy, and then compare the PC strategy with the Chain strategy. We then discuss the segment strategy, its variants (the MOS strategy and the SS strategy), and conclude with the threshold strategy. Finally, we discuss CQ plan characteristics and strategies for avoiding starvation. Section 6.4 presents our quantitative experimental results, detailed analysis, and comparison with theoretical results. We summarize our scheduling work in Section 6.5.

6.1 Scheduling Model and Terminology

In a DSMS, a CQ plan is decomposed into a set of operators (as in a traditional DBMS) such as project, select, join, and other aggregate operators and