**Literature Review:**

**“Scheduling for Overload in Real-Time Systems”,** Sanjoy K. Baruah,

No on-line scheduling algorithm operating in an uniprocessor environment can guarantee to obtain a useful processor utilization greater than 0.25 under conditions of overload. This result holds in the general case, where the deadlines of the input tasks can be arbitrarily “tight.” We address here the issue of improving overload performance in environments where there is a limit on the tightness of task deadlines. In particular, we present a new scheduling algorithm, ROBUST, that efficiently takes advantage of these limits to provide improved overload performance and is asymptotically optimal. We also introduce the concept of overload tolerance, wherein a system’s overload performance never falls below its design capacity, and describe how ROBUST may be used to construct overload tolerant systems.

**“Scheduling to Minimize Staleness and Stretch in Real-Time Data Warehouses”,** MohammadHossein Bateni

We study scheduling algorithms for loading data feeds into real time data warehouses, which are used in applications such as IP network monitoring, online financial trading, and credit card fraud detection. In these applications, the warehouse collects a large number of streaming data feeds that are generated by external sources and arrive asynchronously. Data for each table are generated at a constant rate, different tables possibly at different rates. For each data feed, the arrival of new data triggers an update that appends the new data to the corresponding table; if multiple updates are pending for the same table, they are batched together be- fore being loaded. At time ¿ , if a table has been updated with information up to time. Our ¯rst objective is to schedule the updates on one or more processors in a way that minimizes the total staleness. In order to ensure fairness, our second objective is to limit the maximum \stretch", which we dune (roughly) as the ratio between the duration of time an update waits till it is finished being processed, and the length of the update.

In contrast to earlier work proving the nonexistence of constant-competitive algorithms for related scheduling problems, we prove that any online non preemptive algorithm, no processor of which is ever voluntarily idle, incurs a staleness at most a constant factor larger than an obvious lower bound on total staleness (provided that the processors are sufficiently fast). We give a constant-stretch algorithm, provided that the processors are sufficiently fast, for the quasi periodic model, in which tables can be clustered into a few groups such that the update frequencies within each group vary by at most a constant factor. Finally, we show that our constant-stretch algorithm is also constant-competitive (subject to the same proviso on processor speed) in the quasiperiodic model with respect to total weighted staleness, where tables are assigned weights that react their priorities.

**“Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors”,** YU-KWONG KWOK

Static scheduling of a program represented by a directed task graph on a multiprocessor system to minimize the program completion time is a well-known problem in parallel processing. Since finding an optimal schedule is an NPcomplete problem in general, researchers have resorted to devising efficient heuristics. A plethora of heuristics have been proposed based on a wide spectrum of techniques, including branch-and-bound, integer-programming, searching, graph theory, randomization, genetic algorithms, and evolutionary methods. The objective of this survey is to describe various scheduling algorithms and their functionalities in a contrasting fashion as well as examine their relative merits in terms of performance and time-complexity. Since these algorithms are based on diverse assumptions, they differ in their functionalities, and hence are difficult to describe in a unified context. We propose a taxonomy that classifies these algorithms into different categories. We consider 27 scheduling algorithms, with each algorithm explained through an easy-to-understand description followed by an illustrative example to demonstrate its operation. We also outline some of the novel and promising optimization approaches and current research trends in the area.Finally, we give an overview of the software tools that provide scheduling/mapping Functionalities

**“Supporting Streaming Updates in an Active Data Warehouse”,** Neoklis Panos Vassiliadis

Active Data Warehousing has emerged as an alternative to conventional warehousing practices in order to meet the high demand of applications for up-to-date information. In a nutshell, an active warehouse is refreshed on-line and thus achieves a higher consistency between the stored information and the latest data updates. The need for on-line warehouse refreshment introduces several challenges in the implementation of data warehouse transformations, with respect to their execution time and their overhead to the warehouse processes. In this paper, we focus on a frequently encountered operation in this context, namely, the join of a fast stream S of source updates with a disk-based relation R, under the constraint of limited memory. This operation lies at the core of several common transformations, such as, surrogate key assignment, duplicate detection or identification of newly inserted tuples. We propose a specialized join algorithm, termed mesh join (MESHJOIN), that compensates for the difference in the access cost of the two join inputs by (a) relying entirely on fast sequential scans of R, and (b) sharing the I/O cost of accessing R across multiple tuples of S. We detail the MESHJOIN algorithm and develop a systematic cost model that enables the tuning of MESHJOIN for two objectives: maximizing throughput under a specific memory budget or minimizing memory consumption for a specific throughput. We present an experimental study that validates the performance of MESHJOIN on synthetic and real-life data. Our results verify the scalability of MESHJOIN to fast streams and large relations, and demonstrate its numerous advantages over existing join algorithms