1. INTRODUCTION

A real-time database system (RTDB) is often employed in a dynamic environment to monitor the status of real-world objects and to discover the occurrences of “interesting” events [1, 2, 3, 4]. As an example, a program trading application monitors the prices of various stocks, financial instruments, and currencies, looking for trading opportunities. A typical transaction might compare the price of German Marks in London to the price in New York and if there is a significant difference, the system will rapidly perform a trade. The state of a dynamic environment is often modeled and captured by a set of base data items within the system. Changes to the environment are represented by updates to the base data. For example, a financial database refreshes its state of the stock market by receiving a “ticker tape” — a stream of price quote updates from the stock exchange.

To better support decision making, the large numbers of base data items are often summarized into views. Some example views in a financial database include composite indices (e.g., S&P 500, Dow Jones Industrial Average and sectoral sub-indices), time-series data (e.g., 30-day moving averages), and theoretical financial option prices, etc. For better performance, these views are materialized. When a base data item is updated to reflect certain external activity, the related materialized views need to be updated or recomputed as well.
Besides base item updates and view recomputations, application transactions are executed to generate the ultimate actions taken by the system. These transactions read the base data and views to make their decisions. For instance, application transactions may request the purchase of stock, perform trend analysis, signal alerts, or even trigger the execution of other transactions. Application transactions may also read other static data, such as a knowledge base capturing expert rules. Figure 1 shows the relationships among the various activities in such a real-time database system.

Application transactions can be associated with one or two types of timing requirements: transaction timeliness and data timeliness. Transaction timeliness refers to how “fast” the system responds to a transaction request, while data timeliness refers to how “fresh” the data read is, or how closely in time the data read by a transaction models the environment. Stale data is considered less useful due to the dynamic nature of the data.

Satisfying the two timeliness properties poses a major challenge to the design of a scheduling algorithm for such a database system. This is because the timing requirements pose conflicting demands on the system resources. To keep the data fresh, updates on base data should be applied promptly. Also, whenever the value of a base data item changes, affected derived views have to be recomputed accordingly. The computational load of applying base updates and performing recomputations can be extremely high, causing critical delays to transactions, either because there are not enough CPU cycles for them, or because they are delayed waiting for fresh data. Consequently, application transactions may have a high probability of missing their deadlines.