

# Adaptive Execution of Variable-Accuracy Functions \*

Matthew Denny  
U.C. Berkeley, EECS Dept.  
387 Soda Hall  
Berkeley, CA 94720-1776, USA  
mdenny@cs.berkeley.edu

Michael J. Franklin  
U.C. Berkeley, EECS Dept.  
387 Soda Hall  
Berkeley, CA 94720-1776, USA  
franklin@cs.berkeley.edu

## ABSTRACT

Many analysis applications require the ability to repeatedly execute sophisticated modeling functions, which can each take minutes or even hours to produce a single answer. Because of this expense, such applications have largely been unable to directly use such models in queries, with either on-demand or continuous query processing technology. Query processors are hindered in their ability to optimize expensive modeling functions due to the “black box” nature of existing user-defined function (UDF) interfaces. In this paper, we address the problem of querying over sophisticated models with the development of VAOs (Variable-Accuracy Operators). VAOs use a new function interface that exposes the trade-off between compute time and accuracy that exists in many modeling functions. Using this interface, VAOs adaptively run each function call in a query only to an accuracy needed to answer the query, thus eliminating unneeded work. In this paper, we present the design of VAOs for a set of common query operations. We show the effectiveness of VAOs using a prototype implementation running financial queries over real bond market data.

## 1. INTRODUCTION

### 1.1 Motivation

Many important applications require the repeated use of expensive analysis functions. For example, power companies use models that predict power usage based on variable inputs such as weather conditions. These companies need to run queries using analysis functions to determine the weather conditions that would cause different parts of their grids to become overloaded [5]. As another example, consider securities traders who use numerical models to price securities based on market data. In order to monitor contin-

\*This work was funded in part by NSF under ITR grants IIS-0086057 and SI-0122599, by the IBM Faculty Partnership Award program, and by research funds from Intel, Microsoft, and the UC MICRO program. Denny was supported in part by the Siebel Scholars Fellowship.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '06, September 12-15, 2006, Seoul, Korea.

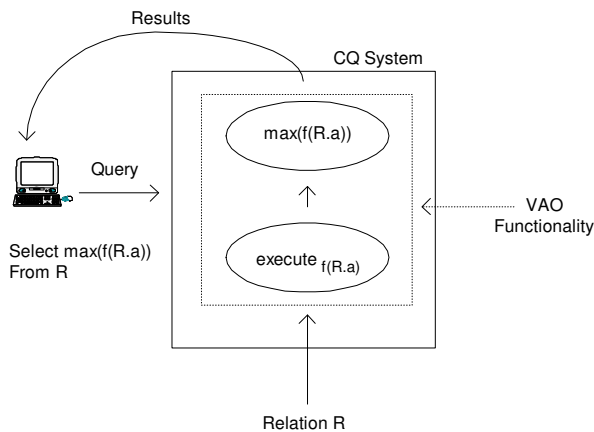
Copyright 2006 VLDB Endowment, ACM 1-59593-385-9/06/09.

uous query results involving these prices, traders must re-run the models as the underlying market data changes. A further example of such applications is in the area of supply chain management (SCM), where users will soon be able to run inventory replenishment models in real-time in response to data provided by emerging RFID technology [12].

Unfortunately, such sophisticated applications often have serious performance problems. In many of these applications, the models require minutes or even hours to compute a single data point, even on modern processors (e.g. [28, 11]). Thus, function execution can easily become a bottleneck. For streaming systems where the function arguments are data streams, the problems are exacerbated; A system may not have the processing power to keep up with the incoming stream updates. At present, analysts have to choose between using less complex (and hence, less accurate) models and running the models less frequently. Neither option is optimal.

If these functions are run in the context of a query, the query processor may be able to reduce the compute cost. While today’s query processors attempt to avoid expensive function calls by either predicate re-ordering [23, 3, 22, 21] or caching [20, 7], such work fails to address the remaining problem of optimizing the *execution* of the function calls that still have to be run. As a result, the applicability of those solutions is limited. In this paper, we address this problem via a new query processing approach called VAOs (Variable-Accuracy Operators). VAOs are based on the insight that many modeling functions, (such as those implementing certain numeric functions) allow a trade-off between compute time and accuracy. VAOs are built upon a new interface to User Defined Functions (UDFs) that provides the system with finer-grained control over function execution, and thus more opportunity for optimization. That is, in contrast to current systems, where UDFs export a “black box”, all-or-nothing interface, VAOs are able to adaptively vary the compute time in functions where if more work is applied, a more accurate answer is obtained.

VAOs perform common query operations (e.g. predicate evaluation, aggregates) that require the execution of expensive functions. In a query plan, VAOs replace both the module that executes a function, as well as the operator that evaluates the result. For the example query plan in Figure 1 for a MAX aggregate over a function result, a single VAO would replace both the function execution and aggregation modules shown. Using the new UDF interface, the VAO adaptively adjusts the amount of work done by the function according to the accuracy needed by the given operation.



**Figure 1: An example system running a simple MAX query. A MAX VAO combines the function execution with the aggregate calculation, enabling adaptive, incremental execution of the expensive function.**

In our figure, the VAO needs to allocate work so that it accurately determines the largest value produced without performing unneeded work on function executions that ultimately produce smaller values. As we will show later, these VAOs often yield drastic performance improvements.

## 1.2 Example Application

To both demonstrate the need for VAOs and illustrate the VAOs approach, we detail the bond trading application from above, which we will use as a running example throughout the paper. As mentioned above, traders often use *bond models* to find a price for a given bond. For bonds that do not trade on an open market, pricing data is often not made public, and traders must use models to obtain prices. These models output a bond price based on input data about the bond and current economic data such as interest rates. As economic and bond data changes, bond traders may want to run models on each bond in real-time, and answer queries such as:

- Q1: Find all bonds priced above \$100.
- Q2: Find the value of my bond portfolio, which is a weighted sum of bond prices.
- Q3: Find the best performing (i.e. highest valued) bond.

In these queries, models must execute quickly because traders need to run a model for each bond issue each time an input changes. In the case of interest rate inputs, the rate is typically calculated using the price of a U.S. Treasury Bond, which changes every 2 minutes on average<sup>1</sup>. Therefore, models must be run quickly in order to be practical.

Unfortunately, bond models such as [11] can be computationally expensive, requiring time on the order of minutes or more. These models require numerical solutions for partial differential equations (PDEs) that cannot be solved analytically. These numerical PDE solvers return approximate prices, where the accuracy of the resulting price depends on the amount of compute work used in the solver.

<sup>1</sup>Determined by observing real-time interest rate data on [30] from 10/13/04 to 11/09/04.

While we concentrate on bond models in this example, similar PDE solvers are used in fields as varied as fluid mechanics [29], semiconductor process design [13], and high-energy physics [24]<sup>2</sup>.

Similar to the processing shown in Figure 1, queries Q1-Q3 can be run in CQ engines with the models supplied as UDFs. Given this architecture and “black box” UDF interface, however, CQ engines cannot control the accuracy of UDF calls. Therefore, these engines must always run models so all answers are accurate enough to answer *any* query. In financial applications, this means running all models with an error of less than \$.01. Since prices can only be accurate to \$.01 anyway, models can effectively report a price as a single real number.

In many cases, these systems often do too much work to process a query. For instance, consider a system running query Q3 over 2 bonds which are worth \$105 and \$95, respectively. Suppose that a model call reports both bond values within \$.01 accuracy. In this case, the system could determine the max value without running the lower valued bond to as high of accuracy, thus requiring much less work.

## 1.3 Overview

To deal with this problem, we present VAOs, which are operators that combine the function execution and the operations over the results. By combining these two operations, the VAO can change the function execution based on the operations performed on the results.

VAOs use a UDF interface which lets them control the work-accuracy trade-off inherent in many functions. Using this interface, functions return upper and lower bounds, not single values. The initial bounds from a function are initially very coarse, as they result from the minimal amount of compute work for the function. If the bounds are not accurate enough to produce an answer for the operator, the VAOs can use the new interface to refine the bounds, which also requires more CPU cycles. A wide variety of numeric functions have an inherent work-accuracy trade-off, and we have modified a variety of numeric algorithms to accommodate the VAO interface.

For a VAO MAX operator in the above example, suppose the functions provide initial bounds of [\$98, \$110] and [\$90, \$101]. Since these bounds overlap, the operator must refine the bounds so that a) maximum value is found, and b) the value is within a certain error tolerance (e.g. \$.01 for bonds). As the VAO can make refinements over either (or both) bounds, each VAO needs a refinement strategy that attempts to conserve work by considering both query operation the data involved.

We have designed VAOs for selection predicates and 4 aggregates. To evaluate our designs, we implemented prototype VAOs and ran experiments using real bond data and models. Under realistic market conditions, these experiments show that VAOs run functions up to two orders of magnitude faster than traditional operators. In additional experiments on synthetic data, we found that VAOs are robust in many experiments explicitly designed to stress VAOs.

<sup>2</sup>While a survey of PDE solvers can be found in Chapter 12 of [2], we discuss PDE solvers in more detail in Sections 2 and 4.



















