Chapter 12:
Panta Rhei: Flexible Execution Engine for Search Computing Queries

Daniele Braga, Stefano Ceri, Francesco Corcoglioniti, and Michael Grossniklaus

Dipartimento di Elettronica e Informazione, Politecnico di Milano,
Piazza Leonardo da Vinci 32, 20133 Milano, Italy
{braga, ceri, corcoglioniti, grossniklaus}@polimi.it

Abstract. The efficient execution of data-intensive computations over services is a challenging task: data are retrieved from remote sources and therefore are not available in the query engine until after the execution of these calls, but the system must be inherently efficient thereafter, by guaranteeing that data is immediately cached and processed efficiently, according to the best query plan. In this chapter, we present a flexible execution model for search computing queries, named Panta Rhei. The proposed execution engine paradigm adopts the producer/consumer model and supports both data-driven and event-driven synchronization, and their interplay. Query plans are modeled as directed graphs, whose nodes are processing units and whose edges are either control or data flows. While control flows synchronize service calls and unit execution, data flows transfer data between units that process data flows to produce query results. We present the specification of Panta Rhei by formally defining the units for data production, consumption, manipulation, and caching, as well as the control and data flows. Finally, we discuss how a query plan is expressed in terms of a query execution plan.

1 Introduction

Query execution in Search Computing is a data-intensive process. The computations required for answering a query, although performed upon the data resulting from service calls, are very similar to those performed by database management systems working on physically optimized tables. Therefore, a query execution engine supporting Search Computing must be able to efficiently support dynamic data extraction, storage and caching, as well as efficiently route data flows between special-purpose computational units, whose design has been optimized so as to guarantee the fast production of query results.

Due to the very nature of many of these tasks and their embedding within Web-based contexts, which are subject to continuous change, performances of data-intensive service interactions are very hard to predict. Moreover, the execution engine must be strongly connected to the query user interface, so as to adapt to user requests that dynamically alter the query requirements, either by specializing current requests or by adding new requirements. For these reasons, the design of the query execution engine for Search Computing has required several architectural solutions for supporting dynamic adaptation which are quite original, especially for what concerns the synchronization aspects.
The main operation in Search Computing is the join of search services and, therefore, the execution engine is optimized to support joins, under the constraint that join data operands are not immediately available to the execution engine, but are produced by interacting with services, ranked and separated in chunks. Join processing, as explained in the previous chapter, aims at exploring given compositions of chunks returned by the services. In this setting, optimization consists in minimizing the number of service calls and, at the same time, in efficiently exploring the search space so as to rapidly produce results.

Supporting join executions requires synchronizing pairs of services. To effect this synchronization, we introduce particular units, called clocks, whose effect is to give pulses to services so as to synchronize them according to certain mutual relationships that can be dynamically adapted. In order to respond to variability, synchronization is subject to feedbacks which are generated within the execution environment. The explicit (and user controllable) synchronization and adaptation of join computations through clock units is the most significant (and original) aspect of the execution engine, being used both for pipelined and parallel execution with a uniform style.

Original aspects of the execution engine concerns the explicit management of chunks within the data flow, which is at the basis of the design of both the chunker units (capable of changing the size of chunks along the data flow) and the cache units (which store the results of service calls by chunks). In SeCo joins, a given chunk of a service’s results can be involved in many chunk combinations, performed after its initial loading, and cannot be discarded until query processing is completed. Chunk support allows for an intermediary granularity level, which is a good compromise between tuple-level (each tuple flows individually) and table-level (each data collection or table flows as a unit) granularity. We believe that this solution yields to a good trade-off between flexibility, adaptability, and performance.

While clocks and chunks are, therefore, the main ingredients of the flexible execution engine, many other features characterize its design. The system must, of course, support sorting (i.e. ranking of results) which is a critical operation, because it is “blocking” (in order for the sort to be applicable to a given collection, all the items of the collection must be available) and data flow machines must try to minimize blocking operations. In addition, the system should support the early evaluation of selection predicates in order to reduce the size of data flows.

The organization of this chapter is as follows. In Section 2 we present the state-of-art of data-driven execution engines, first by highlighting the issues which arise in interpreted environments (such as ours) and then by focusing on adaptability of computations, the main quality offered by Panta Rhei. Section 3 presents the model, with its nodes representing units and edges representing data and control flows. Then, Section 4 sketches the translation of query plans into query engine execution plans, and Section 5 shows the typical translations of parallel, pipe, and top-k joins into schedules.

2 State of the Art

This section gives an overview of the state of the art of query execution with a focus towards the domain of Search Computing. First, we discuss different query processing