Composing, optimizing, and executing plans for bioinformatics web services

1 Introduction

There exist a large number of bioinformatics datasets on the web in various formats. There is a need for flexible and efficient approaches to integrate information from these datasets. Unlike other domains, the bioinformatics domain has embraced web standards, such as XML and web services. A web service is a program that can be executed on a remote machine using standard protocols, such as WSDL and SOAP. There exists a large number of bioinformatics data sources that are either accessible as web services or provide data using XML. For the bioinformatics data sources that provide their data as semi-structured web or text documents, we can use wrapper-based techniques [1–3] to access the data. Most of the available bioinformatics web services are information-providing services, i.e., these services do not change the state of the world in any way. For example, when a user queries the UniProt1 website for details of a protein, the user provides a uniprotid and gets back the information about the protein. Sending this request does not result in side effects, such as charges to the user’s credit card. The emergence of the large number of information-providing services has highlighted the need for a framework to integrate information from the available data sources and services.

In this paper, we describe our approach to automatically compose integration plans to create new information-providing web services from existing web services. When our framework receives a request to create a new web service, it generates a parameterized integration plan that accepts the values of the input parameters, retrieves and integrates information from relevant web services, and returns the results to the user. The parameterized integration plan is then hosted as a new web service. The values of the input parameters are not known at composition time. Therefore, the parameterized integration plan must be able to handle different values of input parameters. This is the key challenge in composing plans for a new web service. To further clarify this, consider the example shown in Fig. 1. We have access

---

1 http://www.pir.uniprot.org/.
to three web services, each providing protein information for different organisms. We would like to create a new web service that accepts the name of an organism and the id of a protein and returns the protein information from the relevant web service. Given specific values of the input parameters, traditional data integration systems can decide which web service should be queried. However, without knowing the values of the parameters, the traditional integration systems would generate a plan that requires querying all three web services.

The key contribution of our approach is to extend the existing techniques to generate parameterized integration plans that can answer requests with different sets of values for the input parameters. This is similar to the problem of generating universal plans [4] in that the generated plan must return an answer for any combination of valid input parameters.

A key issue when generating parameterized plans is to optimize the plans to reduce the number of requests sent to the existing data sources. The existing optimization techniques utilize the constants in the user query to filter out unnecessary source requests and/or reorder the joins to produce more efficient plans. However, as we show with a detailed example later in the paper, those techniques are not enough when we apply them to the task of optimizing parameterized integration plans. Intuitively, we can improve the performance of the parameterized plans for the composed web services using two approaches: (1) by reducing the number of requests sent to web services and (2) by executing requests to the existing web services more efficiently. To that end, we describe two optimizations to reduce the response time of the composed web services: (1) a tuple-level filtering algorithm that optimizes the parameterized integration plans by adding filters based on the source descriptions of the existing web services to reduce the number requests made to the existing web services and (2) an algorithm to map the parameterized integration plans into dataflow-style, streaming execution plans that can be executed efficiently using a highly parallelized, streaming execution engine.

This paper builds on our earlier work, which presented preliminary results on tuple-level filtering [5, 6] and mapping datalog into streaming, dataflow-style execution system [7]. This article describes these techniques in more detail, shows how they can be applied to the bioinformatics domain, and contains new experimental results on real-world bioinformatics web services.

We begin by describing a motivating example that we use throughout the paper to provide a detailed explanation of various concepts. Next, we discuss how existing data integration techniques can be extended to model web sources as data sources and reformulate web service creation requests into parameterized integration plans. Next, we describe an optimization technique termed tuple-level filtering that introduces filters and sensing operations in the parameterized integration plan to reduce the number of requests to the existing web services. In addition, we present a discussion on the applicability of the tuple-level filtering in the bioinformatics domain. Then, we describe techniques to translate recursive and non-recursive datalog composition plans into integration plans that can be executed by a dataflow-style execution engine. Our experimental evaluation shows that the techniques described in this paper achieve a significant reduction in the response time of the composed web services. We conclude the paper by discussing the related work, contributions of the paper, and future work.

2 Motivating example

In this section, we describe a set of available web services and an example web service that we would like to create by composing the available services. The existing web services provide information about various proteins and interactions between different proteins. We model each web service operation as a data source with binding restrictions. The ‘$’ before the attribute denotes that the value for the attribute is required to obtain the rest of the information, i.e., the attribute is a required input to the web service operation. Each data source provides information about one or more domain concept(s). A domain concept refers to a type of entity, e.g., protein.

As shown in Table 1, we have access to eight different web services that provide information about various proteins. Six of these web services, namely, HSProtein, MMProtein, MembraneProtein, TransducerProtein, DIPProtein, and ProteinLocations provide information about proteins. The HSProteinInteractions and MMProteinInteractions services provide information about interactions between proteins.

The HSProtein, MMProtein, MembraneProtein, and TransducerProtein services accept the id of a protein and provide the name of the protein, the location of the protein...