A Service-Based Architecture for Multi-domain Search on the Web

Alessandro Bozzon, Marco Brambilla,
Francesco Corcoglioniti, and Salvatore Vadacca

Dipartimento di Elettronica e Informazione - Politecnico di Milano
via Ponzio, 34/5 - 20133 Milano, Italy
{bozzon,mbrambil,corcoglioniti,vadacca}@elet.polimi.it

Abstract. Current search engines lack in support for multi-domain queries, i.e., queries that can be answered by combining information from two or more knowledge domains. Questions such as "Find a theater close to Times Square, NYC, showing a recent thriller movie, close to a pizza restaurant" have no answer unless the user individually queries different vertical search engines for each domain and then manually combines results. Therefore, the need arises for a special class of search applications that combine different search services. In this paper we propose an architecture aiming at answering multi-domain queries through composition of search services and we provide facilities for the execution of multi-domain queries and the visualization of their results, at the purpose of simplifying the access to the information. We describe our service-based architecture and the implemented optimization and distribution options, and we evaluate the feasibility and performance of our approach.

1 Introduction

Throughout the last decade, search has become the most adopted way to access information over the Internet. User queries are becoming more and more engaging for search engines: the single query itself becomes more complex (both in terms of amount and extension of queried information), the user interaction assumes the form of a process instead of a single query and search sessions tend to become longer and focus not only on documents but also on structured objects. The information to be retrieved is often hidden in the so called "deep Web". While vertical domain-specific search engines provide access to this information, a whole class of queries spanning multiple domains, possibly covered by distinct vertical engines, is unsupported and requires the manual intervention of the user.

We define *search computing applications* [6] the new class of applications aimed at responding to multi-domain queries, i.e., queries over multiple semantic fields of interest, by helping users (or by substituting to them) in decomposing queries and manually assembling complete results from partial answers, provided by domain-specific search services.

The motivation for a novel approach to search service composition is due to: the peculiar characteristics of search services (i.e., ranking and chunking of

P.P. Maglio et al. (Eds.): ICSOC 2010, LNCS 6470, pp. 663–669, 2010.
© Springer-Verlag Berlin Heidelberg 2010

results), the data-intensive essence of the orchestration, the ad-hoc optimization strategies of query plans, the algorithms for the computation of composite results (based on different join strategies), and the novel user interaction paradigms that let users access queries and interact with results. These aspects also challenge the performance and scalability issues of traditional SOA solutions.

2 Architecture

In this section we present an overview of the reference architecture (see Figure 1) of a search computing system, and we provide some details of the main modules. Components are partitioned in three deployment environments (*client*, *server* and *services*) organized as on-line (see Section 3.1) and off-line (see Section 3.2) components.

The search infrastructure is built on top of search services, which may consist of wrapped websites, relational databases, Web services, ReST APIs, SPARQL endpoints, or any other data resources.

Data produced by search services are extracted and combined according to a query execution plan. The query processor accepts users' queries as input, defines the execution plan and orchestrates the invocation of services. The orchestration is performed according to user and service statistics collected and analyzed by means of an off-line profiler. When needed, service materialization increases the overall performance of the system, thanks to local replication.

Finally, a set of tools allows administrators to manage the system and to configure search applications. Caching techniques and the use of multiple query processor instances, together with a load balancer, increase availability and performance of the overall system.

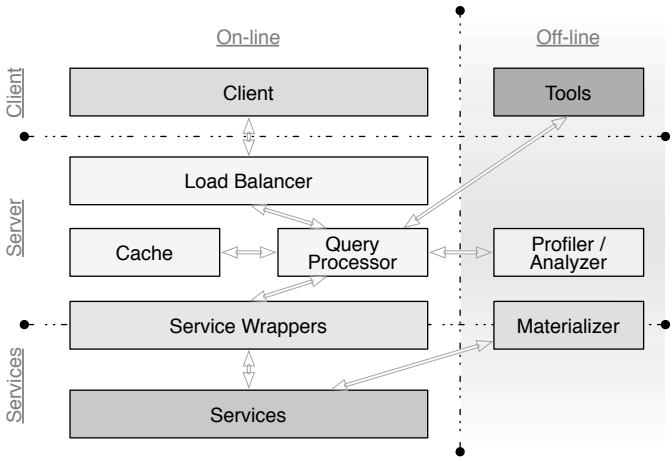

Fig. 1. Overview of the Search Computing architecture

2.1 Client Tier

The client tier comprises the Search Computing user interface. It is based on the *Liquid Query* [2] approach, a "search as a process" information seeking paradigm based on two distinct interaction steps: the initial query submission and the result browsing and query refinement.

At *initial query submission* time, the user specifies the actual parameters of his search, either by filling up a search form or by expressing her needs via natural language queries. The result set computed by the server is shown by the client according to the user's data visualization choices.

Result browsing and query refinement options are offered as a set of interaction primitives that support manipulation, exploration and expansion of the search results, thus allowing for continuous evolution of the query and of the result set itself. The user can also apply data-driven operations to results such as: attribute projection, filtering, sorting, grouping, clustering, and so on. Finally, users can switch between alternative data visualization options such as maps and graphs.

The client is designed as a Rich Internet Application (RIA) architecture running in a Web browser, providing local data storage and manipulation, off-line functioning, and asynchronous, bidirectional client-server communications. Application designers can configure the default data visualization and interaction paradigm, leaving to end users the possibility to dynamically select and download additional visualization components providing alternative views of result sets.

2.2 Server Tier

The Server side mainly comprises the Query Processor, plus a set of caching and balancing facilities. The Query Processor is the component devoted to the management and orchestration of users' queries. The component is accessed by external users and tools through a ReST API and comprises the following modules.

Query Analyzer and Optimizer. It translates user-specified queries into an internal format, which is then optimized by the Query Optimizer [4] according to expected invocation costs and intermediate and final result sizes, as calculated by the offline Profiler/Analyzer. Eventually, the optimization process produces a query plan; query optimization can take place on-line (providing optimized plans in response to user queries) and off-line (refining recurring plans for reuse by other users).

Execution Engine. It executes Panta Rhei query plans [3]. The role of the engine is to produce an execution environment where results from service calls flow within the engine to generate new result combinations at the maximum speed, at the same time also supporting adaptation to changes in the search process and to other runtime factors. The system provides users with both synchronous (pull) and asynchronous (push of results to the user) search mechanisms and fine-grained control to allow for interaction with the search process, in order to dynamically orchestrate it (e.g., to react to the results being produced).

Query Registry. The aim of the Query Registry is to store optimized query execution plans for future reuse.

Mart Registry. We define the abstraction of a search service by means of the notion of Service Mart [5]. The purpose of the Mart Registry is to store *Service Marts*, with expected attributes and their data types; *Connection Patterns*, introducing pair-wise couplings of service marts that define possible join paths; and pointers to the concrete *Service Interfaces* that implement the marts, together with the profiling information collected by the analyzer component.

Service Invoker. When invocations to search services are requested, the invoker looks up the Mart registry and, based on the information at hand, performs the physical invocation of services, possibly through service wrappers.

2.3 Services Tier

While Services consist of the actual search service implementations made available by third party providers, Service Wrappers are components that expose a homogeneous interface to search services, providing retrieved data in a common format processable by the higher layers of the architecture.

In the picture, the component is half-way between the server and the service side to represent two different usage scenarios: the wrappers can be deployed either at the service provider premises or at server side within the Search Computing environment.

3 Distribution Issues

The proposed architecture could be easily compared to a distributed Web retrieval system [1]. We classify the operations executed within the system in the two typical classes of *on-line processing* of queries and *off-line indexing and analysis* of user and service data and metadata. Both on-line and off-line aspects require hardware and software able to cope with high data volumes and to handle high query throughput. Indeed, the number of existing sites and services and the rapidly growing amount of user generated content pose several challenges to the system in terms of result quality and scalability.

3.1 On-Line Query Processing

The system allows users to submit multi-domain queries by means of the Liquid Query UI or by exploiting the available ReST API.

The architecture takes advantage of caching mechanisms. Cache servers hold results of frequent and popular queries, thus reducing the load of the query processor and the latency of the query. Partial results obtained after service calls are also stored in cache. This allows shortening the query latency whenever the execution plan partially overlaps with a previously executed one. Since service invocations are the bottleneck of the architecture, a careful planning and organization of the cache boosts the performance of the overall system, reduces the

bandwidth requirements and also increases the availability and fault tolerance of the infrastructure. A load balancer assigns queries to processors according to the availability of the nodes and their internal load. In the future we also plan to take into account geographical position of clients, nodes, and services, so as to route queries to the appropriate query processor according to the geographical proximity of the client to the processor node and the distance of the processor node from the service provider. The architecture exploits a set of registries to store execution plan descriptions and metadata of the services. The system currently supports several distributed stores; their adoption allows several instances of the query processor to share the data, taking advantage of the partitioning and replication features of the underlying back-end.

3.2 Off-Line Data Indexing and Analysis

User and service profiling is a time-consuming and computation-intensive task, which must be performed off-line and requires ad-hoc hardware and software. Profiling improves the user experience and the efficiency, as we describe below.

User profiling. The user is part and parcel of the search process. He submits queries, expands results, chooses the combinations which best fit to her needs and expresses her preferences. Tracking users' behaviors allows providing high-quality answers: the query execution plan comprises user's favorite services, ranking of result combinations is defined on a per-user basis [8] and result diversification [7] is promoted to address the interests of the user.

Service profiling. Services are subject to a variable load in the space of a day, according to the provenance of users accessing it. If services are correctly profiled, the system can apply more precise load balancing, by querying services with lower load. Moreover, data coming from different services may not be updated with the same frequency. If the update frequency is profiled, ad-hoc time-to-live values can be fixed for the cached data in order to avoid stale data and maximizing the use of the cache at the same time.

Data prefetch. The system implements some proactive behavior by anticipating possible user needs. Recurring usage patterns can be identified and some tasks can be anticipated. The system can decide either to show prefetched data anticipating the user input or to react to the user input with prefetched data.

Service materialization. Service materialization is a bulk operation that is useful whenever there is the need for invoking a service with heavy load. The a-priori download of its data is the best way to provide the answer in a reasonable amount of time. When materialization is performed, the Search Computing infrastructure exposes its own surrogate of the service, guaranteeing better performance due to the locality and replication of data.

4 Experimental Evaluation

In this section we evaluate functionalities and performance of our system. We consider the scenario in which the user is looking for a cinema displaying a thriller movie near Times Square in New York City and a pizza restaurant nearby.

A prototype, together with a few other examples in different scenarios, is available at the following URL: `http://demo.search-computing.com/`. In the prototype, movie information is retrieved through the IMDb archive API, showtimes and theaters through Google Movie Search, and restaurants through the Yahoo Local source. The user interface allows customers to submit their search criteria and to get a set of results, that can be further explored by asking for more combinations, expanding to other concepts, and so on.

Service invocation is the most time-consuming task, requiring respectively 0.2, 0.4 and 0.7 seconds for the Theater, Restaurant and Movie services. The first query result is computed after 4.368 seconds. The system suggests as a first result combination the movie "Iron Man 2" at the "Regal E-Walk 13" theater (which is 0.4 miles far from Times Square) and then a pizza at "Times Square Deli" (300 feet far from the cinema). Globally, the system retrieves 308 results in 18.752 seconds, with 15 invocations of Movie, 7 of Theater and 111 of Restaurant.

Figure 2(a) shows the average number of service invocations required to obtain the top-k query results, with k ranging from 1 to 100, averaging out over multiple random query inputs. The number of invocations of the Restaurant service is almost linear in k, since each one produces an average of 10 query results (its chunk size). Fewer invocations of the Movie and Theater services are required since the number of combinations generated after the parallel join of the two is enough to invoke the Restaurant service to produce the desired amount of results. In some cases, an increase of k may also correspond to a lower number of invocations of some services, due to the impact of other services and compositions on the combination number. In the example, this happens for the Movie and Theater services, due to variations in the number of combinations extracted after the parallel join.

Figure 2(b) shows the impact of caching on the average query execution time, for cache hit probabilities ranging from 0 to 1. Experimental results were obtained by testing our prototype with 20 concurrent clients, running the query of

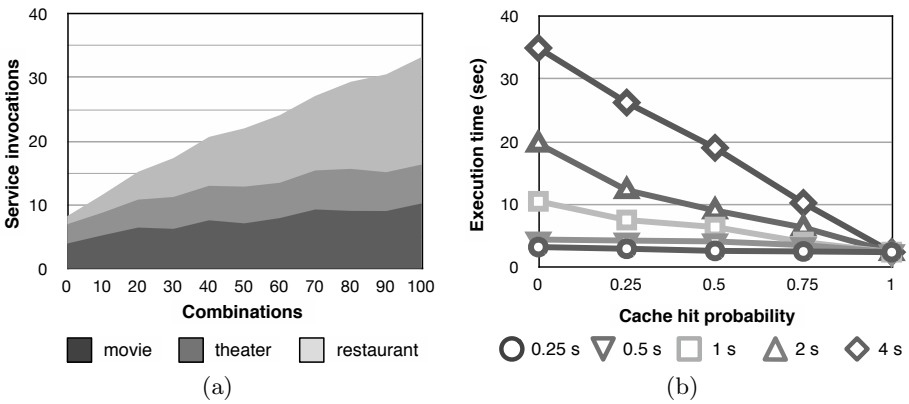

Fig. 2. (a) Required service invocations vs. desired number k of results; (b) Response time vs. probability of cache hit for different average service response times

our scenario to extract the top 10 results from synthetic services whose response time can be configured and ranges from 0.25 sec to 4 sec. Average query execution times dramatically decrease with the growth of cache hit probabilities. Ideally, a cache hit probability of 1 lends to the same execution time regardless of service performance, which depends only on the overhead introduced by our system and, therefore, represents a measure of its efficiency.

5 Conclusion

In this paper we presented a distributed architecture for multi-domain web search applications, according to the Search Computing vision [6]. From a functional standpoint, our approach allows for exploratory search of composite concepts in a service-based environment with no navigational constraints. Our implementation and experimental results demonstrate the feasibility of the approach and show how standard optimization techniques such as distribution, load balancing and caching can be proficiently applied to the multi-domain search problem and are actually needed in a realistic setting to grant performances acceptable for the end user.

Acknowledgments. This research is part of the Search Computing (SeCo) project [www.search-computing.org], funded by the European Research Council. We wish to thank all the members of the SeCo team for their contributions.

References

1. Baeza-Yates, R.A., Castillo, C., Junqueira, F., Plachouras, V., Silvestri, F.: Challenges on Distributed Web Retrieval. In: 23rd International Conference on Data Engineering, ICDE 2007, Istanbul, April 15-20, pp. 6–20 (2007)
2. Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P.: Liquid Query: Multi-domain Exploratory Search on the Web. In: WWW 2010: Proceedings of the 19th international conference on World wide web, pp. 161–170. ACM, New York (2010)
3. Braga, D., Ceri, S., Corcoglioniti, F., Grossniklaus, M.: Panta Rhei: Flexible Execution Engine for Search Computing Queries. In: Ceri, S., Brambilla, M. (eds.) Search Computing Challenges and Directions. LNCS, vol. 5950, pp. 225–243. Springer, Heidelberg (2010)
4. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of Multi-domain Queries on the Web. PVLDB 1(1), 562–573 (2008)
5. Campi, A., Ceri, S., Maesani, A., Ronchi, S.: Designing Service Marts for Engineering Search Computing Applications. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 50–65. Springer, Heidelberg (2010)
6. Ceri, S., Brambilla, M. (eds.): Search Computing. LNCS, vol. 5950. Springer, Heidelberg (2010)
7. Gollapudi, S., Sharma, A.: An Axiomatic Approach for Result Diversification. In: Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, pp. 381–390 (2009)
8. You, G., Hwang, S.: Personalized Ranking: a Contextual Ranking Approach. In: Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, pp. 506–510 (2007)