Avalanche: Putting the Spirit of the Web back into Semantic Web Querying

Cosmin Basca and Abraham Bernstein

DDIS, Department of Informatics, University of Zurich, Zurich, Switzerland
{lastname}@ifi.uzh.ch

Abstract. Traditionally Semantic Web applications either included a web crawler or relied on external services to gain access to the Web of Data. Recent efforts, have enabled applications to query the entire Semantic Web for up-to-date results. Such approaches are based on either centralized indexing of semantically annotated metadata or link traversal and URI dereferencing as in the case of Linked Open Data. They pose a number of limiting assumptions, thus breaking the openness principle of the Web. In this demo we present a novel technique called AVALANCHE, designed to allow a data surfer to query the Semantic Web transparently. The technique makes no prior assumptions about data distribution. Specifically, AVALANCHE can perform "live" queries over the Web of Data. First, it gets on-line statistical information about the data distribution, as well as bandwidth availability. Then, it plans and executes the query in a distributed manner trying to quickly provide first answers.

1 Introduction

With the rapid growth of the Web of Data, unexplored avenues for application development are emerging. While some application designs include a Semantic Web (SW) data crawler, others rely on services that facilitate access to the Web of Data (WoD) either through the SPARQL protocol or various API's (i.e. Sindice or Swoogle). As the mass of data continues to grow – currently LOD [2] accounts for 4.7 billion triples – the scalability factor will give raise to a new set of challenges. Marginally addressed today is the question: How to query the Web of Data on-demand, without hindering the flexible openness principle of the Web – seen as the ability to query independent un-cooperative semantic databases, not controlling their distribution, their availability or having to adhere to fixed publishing guidelines (i.e. LOD). Currently, some approaches maintain a global index over all SW data, striving to deliver up-to date results. They become expensive as the quantity of data grows and demand rises. Derived from traditional distributed database systems, they offer high performance, but break the flexibility and openness principle by assuming perfect knowledge about participating data. Instance level federation (i.e. subjects are distributed to hosts according to a known scheme) as in the case of SemWIQ [5] relaxes the centralization constraints by increasing the flexibility of the system while still holding some constraints over data distribution. As proposed by Hartig et al. [4] other approaches are based on the principle of link traversal and URI dereferencing. These algorithms are driven by LOD principles, offering high flexibility at the cost of potentially expensive query processing (due to network latency) and the impossibility to issue certain types of queries as pointed out by the authors.

This demo proposes to address the issue of preserving the openness and flexibility of WoD while being able to query it "live". Our motivation lies in the belief that such flexibility is paramount for future development of the Semantic Web – as it was for the WWW. Considering the Web's uncontrollable nature, Avalanche [1] strives to find the *first* K results as fast as possible.

2 Avalanche — System Design and Implementation

The system consists of six major components working together in a parallelized pipeline: the Avalanche *endpoints Web Directory* or *Search Engine*, the *Statistics Requester*, the *Plan Generator*, *Plan Executor* instances, *Plan Materializer* instances and the *Query Stopper* component as seen in Figure 1.

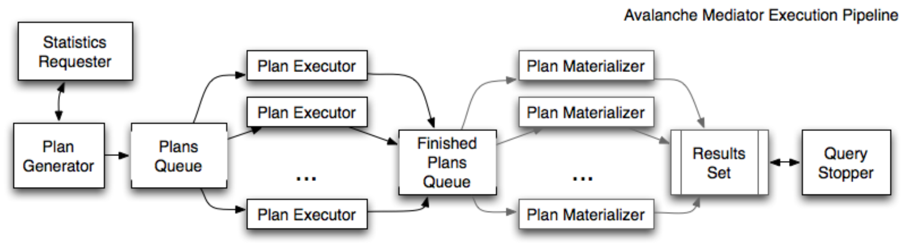

Fig. 1. The Avalanche execution pipeline

The algorithm is comprised of two steps: the *Query Preprocessing* step and the parallel *Query Execution* step. During *Query Preprocessing*, participating hosts are selected via means of a lightweight endpoint-schema inverted index. Ontological prefix (the shorthand notation of the schema – i.e. `foaf`) and schema invariants (i.e. predicates, classes, labels, etc) are appropriate candidate entries to index. After query parsing, this information is immediately available and used to quickly trim down the number of potential endpoints. Further all selected Avalanche endpoints are queried for unbounded variables instance counts.

Query Execution follows: first the query is broken down into the superset of all **molecules**, where a molecule is a subgraph of the overall query graph. A combination of minimally overlapping molecules, covering the query graph, is referred to as a **solution**. Binding all molecules in a given solution to physical hosts that may resolve them, transforms a solution into a **plan**. It is the *Plan Generator's* job to issue plans for execution, as soon as assembled. An emergent challenge from preserving the openness of the query process and the flexibility of semantic data publishing, is denoted by the exponential complexity class of the plan composition space. It is thus crucially important to issue "good" plans first (plans that aid in finding the answers fast) while pruning and stopping undesired plans as soon as possible. To this end, the molecule space is trimmed empirically and plans are generated ordered given their *objective* function. The objective of a plan is represented as the ratio between the *utility* (number of results produced) and the *cost* of execution (time spent to execute subqueries, distributed joins and bandwidth consumption). Consider for example the following query over RDF data describing publications:

```
SELECT ?name ?title WHERE {
?paper akt:has-author ?author; akt:has-author ?jim; akt:has-title ?title.
?author akt:full-name ?name. ?jim akt:full-name "James A. Hendler". }
```

Avlanache's goal is to return the list of all authors that wrote papers with "James A. Hendler" (bound variable) and their titles, given that the required data is spread with an unknown distribution over several independent hosts.

At a given moment during the execution of a plan, a *Plan Executor* may find itself in the following state: molecule `M1` (see Figure 2) was reported to be highly selective on host A, while the remainder of the query (molecule `M2`) is a low selectivity molecule on host B. Given that bandwidth and latency cost can be high, partial results [1] are not sent to one central site to be joined. Instead we start the execution with the highly selective molecule `M1` (host A) and then filter results on host B by sending over results from host A as in Figure 2.

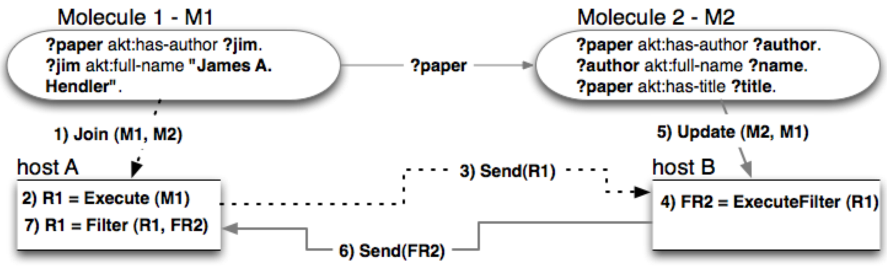
Molecule 1 - M1
?paper akt:has-author ?jim.
?jim akt:full-name "James A. Hendler".
?paper
Molecule 2 - M2
?paper akt:has-author ?author.
?author akt:full-name ?name.
?paper akt:has-title ?title.
1) Join (M1, M2)
5) Update (M2, M1)
host A
2) R1 = Execute (M1)
7) R1 = Filter (R1, FR2)
3) Send(R1)
host B
4) FR2 = ExecuteFilter (R1)
6) Send(FR2)

Fig. 2. Distributed Join and Update operations for a Simple Plan

Similarly the *Plan Materializer*, materializes out-of-order finished plans, by merging partial results and fetching their actual string representations. Since we have no control over the distribution and availability of RDF data and data providers (SPARQL endpoints), getting a complete answer to the query is an unreasonable assumption. Instead the *Query Stopper* monitors for the following *stopping conditions*: a global timeout, a *first K* results policy – all execution stops as soon as K (unique) results are returned to the caller – and finally, to avoid waiting for the timeout when the number of results is $\ll$ K we measure relative result-saturation (i.e. we stop at 90% saturation).

The cost of expensive distributed n-way joins can be reduced via bloom-joins [6]. We extend the objective function with a qualitative join estimation, as described in [3] and since constructing *bloom filters* for large sets is expensive, we only consider the highly selective triple patterns (a threshold set empirically).

[1] It is important to note that to execute plans, hosts will need to share a common id space – a given in Semantic Web via URIs. Naturally, using RDF strings can be prohibitively expensive. To limit bandwidth requirements we, chose to employ a single global id space in the form of the `SHA` family of hash functions on the URIs.

3 Preliminary Evaluation

To demonstrate the Avalanche algorithm and evaluate its performance we conducted a preliminary evaluation on the combined IEEE, ACM and DBLP LOD datasets, totaling 35 million triples. We distributed the datasets over a five-node cluster, splitting by dataset and chronological order (i.e. ACM articles till 2003 on host A). Each machine had 2GB RAM and an Intel Core 2 Duo E8500 @ 3.16GHz. Avalanche was able to successfully execute query plans and retrieve many up-to-date results without having any prior knowledge of the data distribution. Furthermore, we observed that different objective functions have a significant influence on the outcome and should play a critical role when deployed on the Semantic Web. *The demo*[2] *will mirror this setup and let people pose arbitrary queries live.*

4 Conclusion

In order to preserve the openness of the Web of Data, one has to make shallow or no prior assumptions to data distribution and availability. Also given the uncontrollable nature of the Web, a **first K** results policy makes sense. For this purpose we developed and present Avalanche a novel approach for querying the Web of Data that: (1) makes no assumptions about data distribution, availability, or partitioning, (2) provides up-to-date results, and (3) is flexible as it makes no assumption about the structure of participating triple stores. To our knowledge, Avalanche [3] is the first Semantic Web query system that makes no assumptions about the data distribution whatsoever. Whilst, it is a first prototype with a number of drawbacks it represents a first important step towards bringing the spirit of the Web back to triple-stores — a major condition to fulfill the vision of a truly global and open Semantic Web.

References

1. C. Basca and A. Bernstein. Avalanche: Putting the Spirit of the Web back into Semantic Web Querying. In *Proceedings of the 6th International Workshop on Scalable Semantic Web Knowledge Base Systems*, November 2010.
2. C. Bizer, T. Heath, and T. Berners-Lee. Linked data - The story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.
3. A. Broder, M. Mitzenmacher, and A. B. I. M. Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.
4. O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL queries over the Web of linked data. In *8th International Semantic Web Conference (ISWC)*, 2009.
5. A. Langegger, W. Wöß, and M. Blöchl. A Semantic Web middleware for virtual data integration on the web. In *5th European Semantic Web Conference*, 2008.
6. S. Ramesh, O. Papapetrou, and W. Siberski. Optimizing distributed joins with bloom filters. In *ICDCIT '08: Proceedings of the 5th International Conference on Distributed Computing and Internet Technology*, 2009.

[2] Video available at: http://www.ifi.uzh.ch/ddis/fileadmin/basca/avalanche-demo.mov

[3] This work was partially supported by the Swiss National Science Foundation under contract number 200021-118000