Editor: Steffen Staab
University of Koblenz-Landau
staab@uni-koblenz.de

Web Semantics in the Clouds

Peter Mika, *Yahoo! Research*
Giovanni Tummarello, *Digital Enterprise Research Institute*

In the last two years, the amount of structured data made available on the Web in semantic formats has grown by several orders of magnitude. On one side, the Linked Data effort has made available online hundreds of millions of entity descriptions based on the Resource Description Framework (RDF) in data sets such as DBPedia, Uniprot, and Geonames. On the other hand, the Web 2.0 community has increasingly embraced the idea of data portability, and the first efforts have already produced billions of RDF-equivalent triples either embedded inside HTML pages using microformats or exposed directly using eRDF (embedded RDF) and RDFa (RDF attributes).

Incentives for exposing such data are also finally becoming clearer. Yahoo!'s SearchMonkey, for example, makes Web sites containing structured data stand out from others by providing the most appropriate visualization for the end user in the search result page. It will not be long, we envision, before search engines will also directly use this information for ranking and relevance purposes—returning, for example, qualitatively better results for queries that involve everyday entities such as events, locations, and people.

Even though we're still at the beginning of the data Web era, the amount of information already available is clearly much larger than what could be contained, for example, in any current-generation *triple store* (a database for storing and retrieving RDF metadata) typically running on single servers.

Although many applications will need to work with large amounts of metadata, one particular application would certainly not exist without the capability of accessing and processing arbitrary amounts of metadata: search engines that locate the data and services that other applications need. For this reason, Semantic Web search engines and large-scale services are now the first in harnessing grid computing's power when it comes to scaling far beyond the current generation of triple stores.

Cloud computing for the web of data

Not all computationally intense problems require similarly structured hardware configurations. Classic supercomputers, typically characterized by superior arithmetic performance per CPU and high-end CPU interconnection technologies, are proven tools that have scored great successes in physics, astronomy, chemistry, biology, and many other fields. In general, researchers find it hard to surpass such high-end machines when they're facing problems that tend to be hard to parallelize or when they need intense interprocess communication.

When this isn't the case, however, cluster-computing approaches typically exhibit far greater flexibility in resource utilization and much lower overall costs. In an extreme case, clouds can extend to the entire Internet: computations involving relatively small data blocks, with no coordination needs and no significant constraints on execution times, have been performed across the Internet—for example, using free desktop cycles such as in the SETI or Folding@ projects.

The computations needed in Web data processing lie somewhere between these extremes. On the one hand, Web data is interlinked, and the analysis of its mesh has proven to be fundamental in gaining insights about its implicit nature. At the same time, however, the data is by nature distributed and can be said to be consistent with itself, if at all, only within the boundaries of a single Web site. Furthermore, a prominent characteristic is definitely the sheer amount of such data, with a petabyte being a common order of magnitude.[1]

To process this kind of data, leading Internet search providers have been pioneering ways to perform large-scale Web data computations on clusters of commodity machines interconnected with mainstream networking technology. Google's publications about its MapReduce framework[2] and more recently the Yahoo!-initiated, open source implementation Hadoop (www.hadoop.apache.org) are attracting increasing attention from developers and users alike.

The MapReduce style of computation works well with the possibilities offered by the emerging cloud-computing paradigm. This computation style provides generally useful abstractions so that developers can focus on the task at hand (see the sidebar "More about These Technologies"). Executing computations "in the clouds" refers to the model in which an application requests computational resources from a service provider without needing to bother with the

1541-1672/08/$25.00 © 2008 IEEE
Published by the IEEE Computer Society

More about These Technologies

Although distributed computing is almost as old as computer networks, the paradigm has been rapidly gaining popularity recently under the name *cloud computing*. This surge in interest is partly due to limitations of single-machine hardware architectures but is also the result of improvements in software abstractions that hide the complexity of underlying hardware architectures from the programmer. Three of these software layers—Hadoop, HBase, and Pig—all implement data structures (maps and relational tables, for instance) and processing pipelines that are familiar to most programmers. In cases where the developer can map the problem at hand to these solution spaces, the task of cluster programming becomes almost as simple as developing software for single machines (although testing might be more involved). All three are Java-based open-source projects developed under the Apache organization. The descriptions we provide describe only the systems' core functionality.

Hadoop

Hadoop (http://hadoop.apache.org/core) implements a distributed file system (HDFS) and a MapReduce framework similar to the Google File System and Google's original implementation of MapReduce.[1] HDFS provides a virtual disk space that is physically distributed across the machines, where the data is replicated to withstand the failure of single machines and speed up processing. MapReduce provides a simple processing pipeline consisting of two phases, a Map and a Reduce phase. The machines in the cluster work in parallel in both phases, running identical jobs but on different slices of the data. MapReduce builds on HDFS in that the input, output, and intermediate results are stored on the distributed file system.

The input and output of the Map phase is a bag of (key, value) pairs, in which the keys need not be unique. The data on disk can be in any structure as long as the developer can provide a function that loads it into key-value pairs. Keys and values can be arbitrarily complex Java objects as long as they implement the necessary interfaces. At the end of the Map phase, the system collects the pairs with the same key and sends them to separate machines. The Reduce-phase jobs then operate on a bag of pairs with the same key to produce some output. (The output most often also consists of key-value pairs to serve as input for some other processing.) So, the Reduce phase can begin only after all the Map tasks are done. However, this is the only synchronization point and the only interprocess communication, thus greatly simplifying implementation.

HBase

HBase (http://hadoop.apache.org/hbase) builds on Hadoop to provide a virtual database abstraction similar to the original BigTable system.[2] An HBase table always has a column that serves as the key; rows can only be located by this key's value (that is, the table has a single index). Thus, HBase tables can also be considered key-value pairs, where the keys are unique and the value provides the content for the non-key columns. BigTable was originally developed for storing the results of crawling, and HBase preserves some of these characteristics such as the ability to store different versions of cell contents, which can be distinguished by time stamp. HBase doesn't provide a join operation on these tables, only simple lookups by key. HBase tables are stored on the distributed file system in an indexed form to support this.

Pig

Pig (http://incubator.apache.org/pig) also builds on Hadoop but goes a step further toward database-like functionality. A table in Pig is a bag of tuples, in which each field can hold a value or a bag of tuples (that is, nested tables are also possible). Tables in Pig aren't stored on disk in any special form but are loaded by a custom load function the programmer defines. (This has the disadvantage that the data isn't indexed by any key.) PigLatin, Pig's scripting language, is procedural—that is, writing a Pig script is more similar to writing a "query plan" than to writing a query in SQL. Nevertheless, most users find learning PigLatin easy because it provides all the familiar constructs of SQL such as filtering (projection), joins, grouping, sorting, and so on. PigLatin can also be extended using custom functions. Furthermore, it can also be embedded inside Java to provide additional control such as conditionals and loops. PigLatin scripts are translated into a sequence of MapReduce jobs.

References

1. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proc. 6th Symp. Operating System Design and Implementation* (OSDI 04), Usenix Assoc., 2004.
2. F. Chang et al., "Bigtable: A Distributed Storage System for Structured Data," *Proc. 7th Symp. Operating System Design and Implementation* (OSDI 06), Usenix Assoc., 2006, pp. 205–218.

computational offer's details. An example that's recently enjoying popularity is the Amazon Elastic Computing service offer, which allows computing capabilities to be allocated within minutes and increased dynamically as required—for example, to quickly cope with an unexpected peak of visitors. Because MapReduce is agnostic with regard to the actual size of the cluster it runs on, executing Hadoop on such cloud-like infrastructures is an appealing strategy for performing data-intensive computations while minimizing, or optimizing, the upfront infrastructure investment. The combined paradigm is generally referred to as *data-intensive scalable (or super)computing* (DISC).

With respect to the complex tasks involved in processing the Web of data, a DISC approach is for many reasons a natural choice. On the one hand, many tasks, such as crawling, can be performed in a fashion similar to the processing of regular Web content (HTML pages). Semantic data crawling often requires special treatment and dedicated intelligence but doesn't generally differ very much from crawling the HTML Web. Similarly, ranking a Semantic Web source such as sites or data sets on the basis of algorithms similar to PageRank, and

thus efficiently computed over MapReduce, has been proposed. On the other hand, many other data-intensive, batch-processing tasks are needed for specifically addressing the challenges of the Web of data. Examples include large-scale data analysis, cleaning, reasoning, entity recognition and consolidation, and ontology mapping.

Grid computing is certainly useful in some of these tasks. We first show how Yahoo! is building on grid computing using Hadoop to enable the analysis, transformation, and querying of large amounts of RDF data in a batch-processing mode using clusters of hundreds of machines, without apparent bottlenecks in scalability. Next, we show how the Semantic Web search engine Sindice is exploiting Hadoop and related technologies to scale semantic indexing beyond the limits of dedicated cluster environments while reducing cost and complexity.[3]

Batch-processing RDF using Yahoo! Pig

Yahoo! is building on grid computing using Hadoop to enable the analysis, transformation, and querying of large amounts of RDF data in a batch-processing mode using clusters of hundreds of machines, without apparent bottlenecks in scalability. The Yahoo! crawler affectionately named Slurp began indexing microformat content in the spring of this year, and the company recently added eRDF and RDFa to its supported formats. Yahoo! has also innovated in the Semantic Web area by allowing site owners to expose metadata using the DataRSS format, an Atom-based format for delivering RDF data. The Yahoo! SearchMonkey application platform will likely produce a further explosion in the amount of data handled. SearchMonkey developers can create so-called custom data services to extract metadata from existing Web sites or turn APIs into sources of metadata. All the metadata collected by Yahoo! is stored in an RDF-compatible format, so processing it requires the ability to query and transform large amounts of RDF data.

Yahoo! aims to develop data solutions that address wide classes of data management problems and that can easily be adapted to new problems, such as the one posed by SearchMonkey.[4] One of these tools, Yahoo! Pig, simplifies the processing of large data sets on computer clusters by applying concepts from parallel databases.[5] It was originally developed inside Yahoo! Research but has been recently made available as open source under the Apache 2.0 license. Pig natively provides support for data transformations such as projections, grouping, sorting, joining, and compositions. The expressivity of Pig's transformation language is roughly equivalent to standard relational algebra (which also forms the basis of SQL), with the added benefit of extensibility through custom functions written in Java. Pig programmers develop custom code for loading and saving data in other formats into Pig's data model, which again builds on the relational model (bags of tuples) with additional features such as maps and nested bags of tuples.

> Using the Sindice API, it's possible to search for people, places, events, and connections on the basis of semantically structured documents found on the Web.

Scripts written in PigLatin, Pig's native language, are executed on the cluster using the Hadoop framework or Galago (www.galagosearch.org), a tuple-processing engine. In contrast to HBase, Pig can't actually be called a database: processing takes place by iterating through the whole data set (the data isn't indexed and can't be updated), and the results of computation are saved. HBase, however, doesn't offer a query language, only the retrieval of tuples by their index.

We observed that Pig's data model and transformation language are similar to the relational representations of RDF and the SPARQL query language, respectively, so we recently extended Pig to perform RDF querying and transformations. As part of this work, we implemented load and save functions to convert RDF to Pig's data model, created a mapping between SPARQL and PigLatin, and proved that the mapping is complete.

We implemented these results as a new back end for the popular Sesame triple store.[6] Sesame comes with several components for storing and retrieving tuples (including an in-memory implementation, a native disk-based implementation, and a relational-database-management-system-based implementation) but also lets us plug in additional back ends with minimal effort. As a result, Sesame applications can switch transparently to a Pig-based RDF store when scalability requires it without requiring any changes to the application code.

We evaluated our system by performing the common Lehigh University Benchmark (LUBM). We experimented with the first two queries of the LUBM set, varying the number of nodes used for computation and the size of the data.

Figure 1 shows Pig's performance using the same query (the first one in the test set) on different sizes of test data and with varying numbers of nodes. In Figure 2, we show how performance changes as we move from the first, simple query to the second, more complex query. The latter requires five joins instead of one.

Both figures show that asymptotically the execution time is more or less independent of the data set's size and is around 100 seconds. In other words, no matter how large a data set is, given sufficient resources this query can be executed in about 100 seconds. On the other hand, we consider this a lower bound, because it's associated with the fixed costs of allocating nodes and distributing the job. This confirms that—as expected—our solution isn't applicable to online, interactive tasks.

The results also show that the improvement from adding additional nodes depends on whether the execution is balanced. A *balanced execution* is one where each node processes an equal number of blocks—that is, when

$$\left\lceil \frac{b}{2*m} \right\rceil = k$$

where $k \in \mathbb{N}^+$, b is the number of blocks, and m is the number of machines. Between points that result in balanced executions, additional nodes won't improve performance, because performance is determined by the nodes that need to process the larger number of blocks. However, we do observe the expected linear scale-up for balanced execution without any notable communica-

tion overhead, even when scaling up to hundreds of nodes. Furthermore, this holds true for the second, more complex query: execution time increases proportionally with the query, but the general scaling behavior stays the same.

Although we don't report the results in this article, we also experimented with RDF reasoning using a forward-chaining algorithm. Reasoning in this scenario requires executing queries iteratively and adding the results to the data set until no more new triples are produced. Also, this method is easy to extend with rule-based implementations of OWL subsets.

As for the limitations of the approach, Pig processing provides a solution only to offline batch-processing tasks because of the overhead of distributing the job to processing nodes and copying data if necessary. Also, because no index is provided, updates require full parsing. (On the other hand, adding data is as simple as copying it to a directory of the distributed file system.) Despite these drawbacks, a MapReduce-based infrastructure is most likely to offer the best resource utilization in an analytical or research scenario by relying only on commodity hardware and offering a general-purpose computing platform. In our research environment, the same cluster of machines that's used to analyze ad clicks or query logs is used to query and reason with metadata, because both tasks can be fitted to the same paradigm. The system allocates computing nodes dynamically and releases them back to the pool as soon as the job is finished.

Because of their versatility, MapReduce clusters are used in production for large-scale data processing at both Google[2] and Yahoo! (http://developer.yahoo.com/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html). However, given that the software is free and requires no special hardware, this solution is also open to universities that must perform experiments with large amounts of metadata.

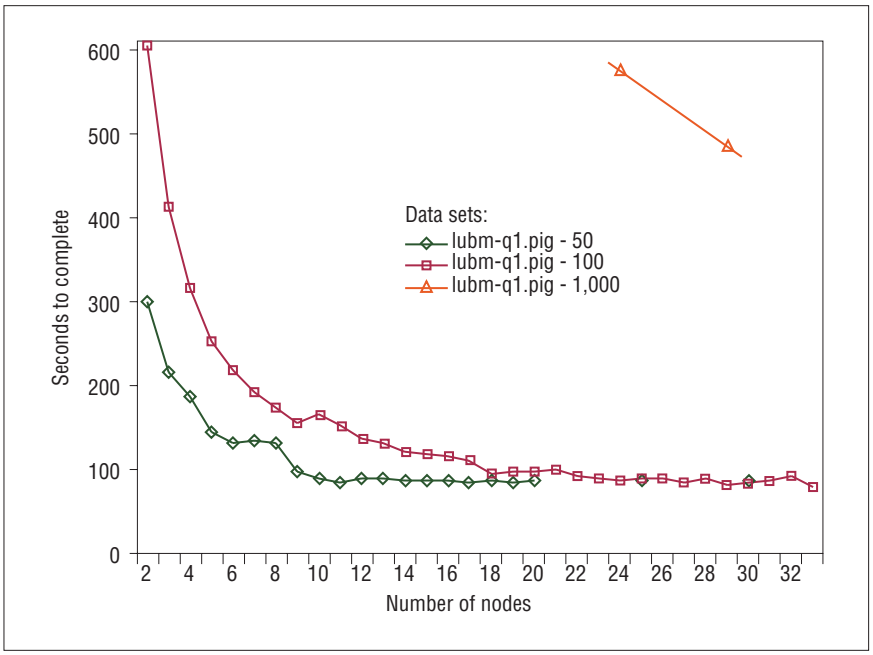

Figure 1. Pig's performance on Query 1 of the Lehigh University Benchmark (LUBM) using varying numbers of nodes. Different curves represent different sizes of data—that is, the LUBM 50, LUBM 100, and LUBM 1,000 data sets.

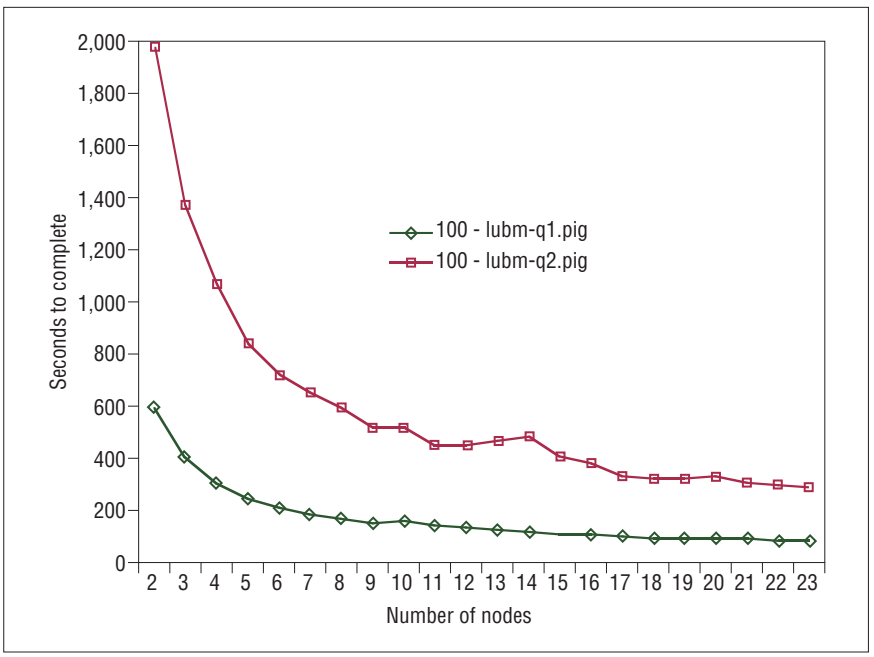

Figure 2. Pig's performance on two queries, both taken from the LUBM benchmark. Q1 is simpler, and Q2 is more complex using varying numbers of nodes.

Sindice: Large-scale processing of structured Web data

Developed by the Digital Enterprise Research Institute's Data Intensive Infrastructures group (http://di2.deri.ie), the Sindice project deals with building scalable APIs to locate and use RDF and microformat data as found on the Web. By using the Sindice API, for example, it's possible to use keywords and semantic-pattern queries to search for people, places, events, and connections on the basis of semantically structured documents found on the Web. This includes, for example, FOAF (friend-of-a-friend) files, HTML pages using the hcard microformat, XML Social Network (XFN) information, and so on.

Technically, Sindice is designed to provide these services, fulfilling three main nonfunctional requirements: scalability, runtime performance, and ability to cope with the many changes in standards and

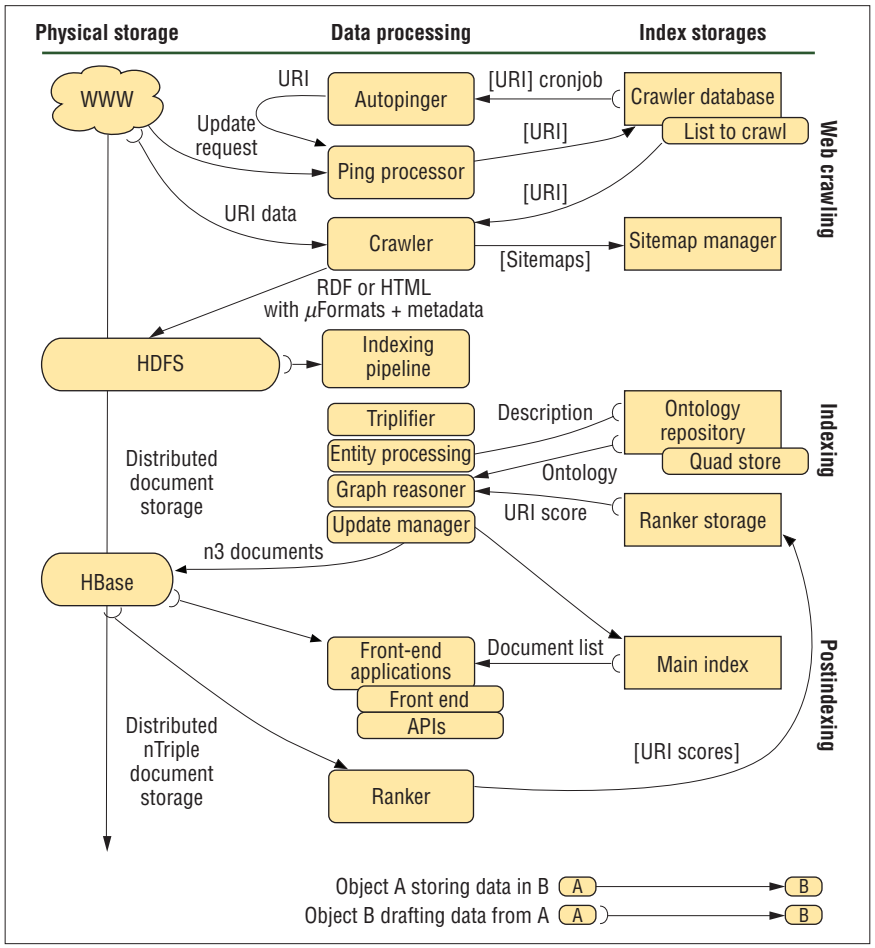

Figure 3. The Sindice architecture. In the Sindice indexing pipeline, documents are first discovered and crawled from the Web. The pipeline then extracts data from the documents and performs reasoning and entity consolidation before creating the index that's used for serving clients.

usage practices on the Web of data. Figure 3 shows a sketch of Sindice's internal architecture.

In terms of scalability, the goal is to handle massive amounts of data on the Web, with predicted scalability needs in the range of trillions of triples. We achieve this by using information retrieval techniques to answer both textual and semantic queries over large collections of structured documents. This approach trades limited query complexity (for example, only allowing simple pattern queries and limited join capability) for high scalability, as we see in traditional Web search engines. Although answering queries doesn't require cloud-computing techniques, these are fundamental for Sindice to reach its scalability goals during the entire preprocessing phase and in particular to perform harvesting, storage, and data transformation.

At the harvesting level, SindiceBot (a Web crawler that collects Web pages and RDF documents for the Sindice search engine) employs Hadoop/Nutch (http://lucene.apache.org/nutch) to distribute the crawling jobs across multiple machines. The main crawling strategy is not unlike what's commonly used in Web harvesting, but specific tasks differ and require data-intensive operations. One such case is the efficient handling of large data sets published using the Linked Data model (a set of best practices for publishing and deploying instance and class data using the RDF data model). DBPedia.org, for example, exposes several million virtual RDF files derived on demand by querying the underlying data set.

Crawling one such site requires considerable time, poses limits to the frequency of recrawling and therefore of information updates by the search engine, and imposes a nonnegligible computational burden on the remote site. This is similar to the problem of Deep Web crawling, which the Sitemap protocol (www.sitemaps.org) is currently addressing. Similarly, an extension to the Sitemap protocol (http://sw.deri.org/2007/07/sitemapextension) allows data sets to be described in a way that the crawler can download the data as a dump and avoid retrieving each individual URI (uniform resource identifier) referenced in the data. Processing such dumps is data intensive, requiring RDF processing at the triple level to create entity representations, which are then individually indexed. Similar to what we described earlier about Pig, the Sindice crawler performs this task efficiently across the cluster using a MapReduce implementation.[7]

After collection, data is stored in its raw form, mostly HTML or RDF, in the HBase distributed column-based store (see the sidebar "More about These Technologies").

For complex data analysis, HBase constitutes the distributed storage medium with possibilities for structured queries—albeit limited—as well as input for any analytics implemented with MapReduce. To support runtime requirements of data analysis tasks, the Sindice index must be precomputed and fully expanded. This is done in three steps.

First, the Sindice indexing pipeline preprocesses raw data from HBase. It analyzes the HTML using the Hadoop pipeline to extract RDFa, microformats, and many other significant elements and summary information. These include statistics but also extra bits of information such as the page's title, the presence of RSS or other machine-processible elements, and context elements such as keywords found in the text. At this point, the document's semantics, regardless of the original format, is represented in RDF and proceeds in the pipeline for reasoning and consolidation.

Second, the Sindice indexing pipeline applies reasoning to fact sets. Reasoning is important when indexing native RDF documents, because it makes information explicit and therefore directly available for indexing purposes. As an example, in the case of social-networking data described using the FOAF ontology, it wouldn't be possible to answer a query for "entities labeled 'giovanni'" unless you could infer that a foaf:name is a subproperty of rdfs:label. Microformats, on the other hand, require

even more sophisticated handling—for example, to extract the named entities from a description text and turn them into triples in the RDF representation. Efficient up-front reasoning requires that MapReduce jobs employ some form of state, which is unusual: MapReduce jobs must be functional by definition, which stands in contrast with the need to reuse previously obtained reasoning results (for instance, reasoning on TBox statements of reused ontologies should be performed just once). Our solution is based on the use of several "reasoning servers" shared across MapReduce jobs, which cache reasoning results in large main memories.

Third, entity consolidation is required to establish appropriate cross-references between the data and its index. Let's assume that the engine has indexed documents containing a given term—for example, URI1. If the engine learns that URI1 owl:sameAs URI2, it will need to reindex all the originally indexed documents to add the equivalence of URI2 in the document term index. Similarly, the indexer often needs to modify the rules for data transformation—for instance, from microformats to RDF—to increase the quality of the search results or to address new usage patterns or new standards. Once this is done, the engine must reindex most of the data starting back from the HTML form.

At the output of the indexing pipeline, the RDF documents are now consolidated and contain explicit semantic statements together with consolidation information. Finally, these are both sent to the index and added to HBase for later retrieval and further offline processing.

Having illustrated the data-intensive processes in a semantic-search engine, the need for distributing storage and effectively sharing the computational load across the cluster is evident. In Sindice, we've found that Hadoop and the associated technology stack effectively address data-intensive tasks, whether for data management or analysis, including those specific to processing structured data. In our experience, the indexing pipeline's processing throughput can grow almost linearly with the number of servers. In the current setup, Sindice's cluster of eight machines can process approximately 150 documents per second.

Cloud-computing techniques, already the processing backbone of the Web of HTML pages, are also now being exploited to deal with the explosion of the Web of data, both in research and in production environments. Hadoop's MapReduce framework and extensions to it such as HBase and Yahoo! Pig are being employed to do large-scale processing of arbitrarily shaped semantic data sets with no scalability limits in sight. As always, conforming to a framework entails the need for adaptation: both data and algorithms must be recast so that they fit platform design. However, in Semantic Web research—and in Web science in general—we can study some of the most interesting phenomena such as the emergence of semantics only on large data sets. So, there's no way to escape the compromises required for addressing problems of scale.

Cloud-computing techniques have been well known for large commercial players, but they're now within the reach even of academic research labs and small and medium-scale enterprises. Thanks to the use of inexpensive, commodity hardware and open source implementations, experimenting with these techniques is easy, even with very low budgets. Adopting cloud-computing techniques, however, requires a special kind of expertise. To aid in this process, cloud computing must have a stable place in academic curricula. Although several universities are starting to provide courses on these paradigms, a notable lag exists between academia and the research centers of large commercial Internet players. Europe, it seems, is also lagging a bit. We can only wish that this theme becomes increasingly recognized as a strategic focus for achieving competitiveness in the Web market.

Finally, with respect to the Semantic Web research community, we are very interested in continuing to develop Semantic Web algorithms cast into the MapReduce framework or its higher-level abstractions such as Pig and HBase. We believe we can successfully transform some of the research problems we've been facing into the well-understood MapReduce paradigm and then apply solutions based on open source implementations and commodity hardware. We call on the research community to explore the entire range of Semantic Web algorithms that could be successfully transformed into this increasingly popular solution space. Many of the Semantic Web's scalability problems will likely turn out to be less challenging after all. ■

Acknowledgments

We acknowledge the support of Ben Reed of Yahoo! Research in carrying out some of the research described. We also acknowledge the support of Renaud Delbru, Michele Catasta, Gabriele Renzi, Paolo Capriotti, Holger Stenzhorn, and Richard Cyganiak for other parts of our research.

References

1. F. Chang et al., "Bigtable: A Distributed Storage System for Structured Data," *Proc. 7th Symp. Operating System Design and Implementation* (OSDI 06), Usenix Assoc., 2006, pp. 205–218.
2. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proc. 6th Symp. Operating System Design and Implementation* (OSDI 04), Usenix Assoc., 2004, pp. 137–150.
3. E. Oren et al., "Sindice.com: A Document-Oriented Lookup Index for Open Linked Data," *Int'l J. Metadata, Semantics, and Ontologies*, vol. 3, no. 1, 2008.
4. R. Baeza-Yates and R. Ramakrishnan, "Data Challenges at Yahoo!" *Proc. 11th Int'l Conf. Extending Database Technology* (EDBT 08), ACM Press, 2008, pp. 652–655.
5. C. Olston et al., "PigLatin: a Not-So-Foreign Language for Data Processing," *Proc. 2008 ACM Sigmod Int'l Conf. Management of Data* (Sigmod 08), ACM Press, 2008, pp. 1099–1110.
6. J. Broekstra, A. Kampman, and F. van Harmelen, "Sesame: An Architecture for Storing and Querying RDF and RDF Schema," *Proc. 1st Int'l Semantic Web Conf.* (ISWC 02), LNCS 2342, Springer, 2002, pp. 54–68.
7. R. Cyganiak et al., "Exposing Large Data Sets with Semantic Sitemaps," *Proc. 5th European Semantic Web Conf.*, LNCS 5021, Springer, 2008, pp. 690–704.

Peter Mika is a researcher at Yahoo! Research. Contact him at pmika@yahoo-inc.com.

Giovanni Tummarello is a research fellow, head of the Data Intensive Infrastructures research unit, and project leader for the Sindice Data Web Search Engine project (http://sindice.com) at the Digital Enterprise Research Institute. Contact him at giovanni.tummarello@deri.org.