L

L1 Cache

► Processor Cache

L2 Cache

► Processor Cache

L3 Cache

► Processor Cache

Language Models

Djoerd Hiemstra
University of Twente, Enschede, The Netherlands

Synonyms

Generative models

Definition

A language model assigns a probability to a piece of unseen text, based on some training data. For example, a language model based on a big English newspaper archive is expected to assign a higher probability to "a bit of text" than to "aw pit tov tags," because the words in the former phrase (or word pairs or word triples if so-called *N-Gram Models* are used) occur more frequently in the data than the words in the latter phrase. For information retrieval, typical usage is to build a language model for each document. At search time, the top ranked document is the one whose language model assigns the highest probability to the query.

Historical Background

The term *language models* originates from probabilistic models of language generation developed for automatic speech recognition systems in the early 1980s [9]. Speech recognition systems use a language model to complement the results of the *acoustic model* which models the relation between words (or parts of words called phonemes) and the acoustic signal. The history of language models, however, goes back to the beginning of the twentieth century when Andrei Markov used language models (Markov models) to model letter sequences in works of Russian literature [3]. Another famous application of language models are Claude Shannon's models of letter sequences and word sequences, which he used to illustrate the implications of coding and information theory [17]. In the 1990s, language models were applied as a general tool for several natural language processing applications, such as part-of-speech tagging, machine translation, and optical character recognition. Language models were applied to information retrieval by a number of research groups in the late 1990s [4,7,14,15]. They became rapidly popular in information retrieval research. By 2001, the ACM SIGIR conference had two separate sessions on language models containing five papers in total [12]. In 2003, a group of leading information retrieval researchers published a research roadmap "challenges in information retrieval and language modeling" [1], indicating that the future of information retrieval and the future of language modeling can not be seen separate from each other.

Foundations

Language models are generative models, i.e., models that define a probability mechanism for generating language. Such generative models might be explained by the following probability mechanism: Imagine picking a term T at random from this page by pointing at the page with closed eyes. This mechanism defines a probability $P(T|D)$, which could be defined as the relative frequency of the occurrence of the event, i.e., by the number of occurrences of a term on the page divided by the total number of terms on the page. Suppose the process is repeated n times, picking one at a time the terms T_1, T_2,...,T_n. Then, assuming independence

between the successive events, the probability of the terms given the document D is defined as follows:

$$P(T_1, T_2,...,T_n|D) = \prod_{i=1}^{n} P(T_i|D) \quad (1)$$

A simple language modeling approach would compute (1) for each document in the collection, and rank the documents accordingly. A potential problem might be the following: The equation will assign zero probability to a sequence of terms unless all terms occur in the document. So, a language modeling system that uses (1) will not retrieve a document unless it contains all query terms. This might be reasonable for a web search engine that typically processes small queries to search a vast amount of data, but for many other information retrieval applications, this behavior is a problem. A standard solution is to use *linear interpolation smoothing* of the document model $P(T|D)$ with a collection model $P(T|C)$, which is defined as follows:

$$P(T_1, T_2, ..., T_n|D) = \prod_{i=1}^{n} \Big(\lambda P(T_i|D) + (1-\lambda)P(T_i|C)\Big) \quad (2)$$

This way, a term that does not occur in the document will not be assigned zero probability but instead a probability proportional to its number of occurrences in the entire collection C. Here, λ is an unknown probability that should be tuned to optimize retrieval effectiveness. Linear interpolation smoothing was used in several early language modeling approaches [7,14].

Implementation

Although the language modeling equations above suggest the need to compute probabilities for all documents in the collection, this is unnecessary in practice. In fact, most language modeling approaches can be implemented efficiently by the use of standard inverted index search systems. This can be seen by the equation below which can be derived from (2) by two basic transformations: First, dividing it by the probability of the collection model; and second, taking the logarithm.

$$P(T_1, T_2,...,T_n|D) \propto \sum_{i=1}^{n} \log\left(1 + \frac{\lambda P(T_i|D)}{(1-\lambda)P(T_i|C)}\right) \quad (3)$$

Equation (3) no longer produces probabilities, but it ranks the documents in the exact same order as (2), because the collection model does not depend on the document, and the logarithm is a strictly monotonic function. Taking the logarithm prevents the implementation from running out of the precision of its (floating point) representation of probabilities, which can become very small because the probabilities are multiplied for every query term. Similar to for instance vector space models in information retrieval, ranking is defined by a simple sum of term weights, for which terms that do not match a document get a zero weight. Interestingly, the resulting "term weight" can be seen as a variant of *tf.idf* weights, which are often used in vector space models.

Document Priors

The equations above define the probability of a query given a document, but obviously, the system should rank by the probability of the documents given the query. These two probabilities are related by Bayes' rule as follows.

$$P(D|T_1, T_2,...,T_n) = \frac{P(T_1, T_2,...,T_n|D)P(D)}{P(T_1, T_2,...,T_n)} \quad (4)$$

The left-hand side of (4) cannot be used directly because the independence assumption presented above assumes term independence given the document. So, in order to compute the probability of the document D given the query, (2) needs to be multiplied by $P(D)$ and divided by $P(T_1,...,T_n)$. Again, as stated earlier, the probabilities themselves are of no interest, but the ranking of the document by the probabilities is. And since $P(T_1,...,T_n)$ does not depend on the document, ranking the documents by the numerator of the right-hand side of (4) will rank them by the probability given the query. This shows the importance of $P(D)$: The marginal probability, or *prior probability* of the document, i.e., it is the probability that the document is relevant if the query is ignored. For instance, it might be assumed that long documents are more likely to be useful than short documents [5,6]. In web search, such a so-called static ranking (a ranking that is independent of the query) is commonly used. For instance, documents with many links pointing to them are more likely to be relevant, or documents with short URLs are more likely to be relevant. The prior probability of a document is a powerful way to

incorporate static ranking in the language modeling approach [10].

Document Generation Models

An implicit assumption of the language models presented is that there is more information available about the documents than about the query. In some applications, however, the situation is reversed. For instance in *topic tracking*, a system has the task of tracking a stream of chronologically ordered stories. For each story in the stream, the system has to decide whether it is on topic. The target topic is usually based on a number of example stories on a certain topic, there is more information available about the topic than about a single story. Unlike query generation models, document generation models need some form of normalization because documents will have different lengths. The probability of generating a document tends to be smaller for long documents than for short documents. Therefore, several normalization techniques might be applied, such as normalization by document length and additional Gaussian normalization [10,18]. *Relevance feedback* (i.e, the user marked some documents as relevant) is another situation in which there is more knowledge available about the query than about each single document. If some relevant documents are known, or if the top ranked documents are assumed to be relevant, then those documents might be used to generate a new, improved query [20]. As an example, consider the following so-called *relevance models* approach [13]

$$P(Q|T_1,\ldots,T_n) \propto \sum_d \left(P(D=d)P(Q|D=d)\prod_{i=1}^{n} P(T_i=t_i|D=d) \right) \tag{5}$$

Here, the formula defines the probability of a new word Q, given the original query $T_1,\ldots,T_n$ by marginalizing over all documents. In practice, only the top ranked documents for the query $T_1,\ldots,T_n$ are used. Interestingly, the relevance model might be used to infer other information from the top ranked documents, for instance the person that is most often mentioned for a certain query, so-called *expert search* [2].

Translation Models

Language models for information retrieval are generative models, and therefore easily combined with other generative models. To add a model of term translation, the following probability mechanism applies: Imagine picking an English term T at random from this page by pointing at the page with closed eyes (which defines a probability $P(T|D)$), and then translate the term T by picking from the term's entry in a English–Dutch dictionary at random a Dutch term S (with probability $P(S|T)$). The model might be used in a cross-language retrieval system to rank English documents given a Dutch query $S_1,\ldots,S_n$ by the following probability [4,6,13,19]:

$$P(S_1,S_2,\ldots,S_n|D) = \prod_{i=1}^{n}\sum_t \Big(P(S_i=s_i|T_i=t) \left(\lambda P(T_i=t|D) + (1-\lambda)P(T_i=t|C)\right)\Big) \tag{6}$$

Here, Dutch is the source language and English the target language. The formula uses linear interpolation smoothing of the document model with the target language background model $P(T|C)$ (English in the example) at the right-hand side of the formula. In some formulations, the translation model is smoothed with the source language background model $P(S|C)$ which a estimated on auxiliary data. The two background models are related as follows: $P(S|C) = \sum_t P(S|T=t)P(T=t|C)$. The translation probabilities are often estimated from parallel corpora, i.e., from texts in the target language and its translations in the source language [6,19]. Translation models might also be used in a monolingual setting to account for synonyms and other related words [4].

Aspect Models

In *aspect models*, also called *probabilistic latent semantic indexing* models, documents are modeled as mixtures of aspect language models. In terms of a generative model it can be defined in the following way [8]: (i) select a document D with probability $P(D)$, (ii) pick a latent aspect Z with probability $P(Z|D)$, (iii) generate a term $T$with probability $P(T|Z)$ independent of the document, (iv) repeat Step 2 and Step 3 until the desired number of terms is reached. This leads to (7).

$$P(T_1,T_2,\ldots,T_n|D) \propto \prod_{i=1}^{n}\left(\sum_z (P(T_i|Z=z)P(Z=z|D)) \right) \tag{7}$$

The aspects might correspond with the topics or categories of documents in the collection such as "health", "family", "Hollywood", etc. The aspect Z

is a hidden, unobserved variable, so probabilities concerning Z cannot be estimated from direct observations. Instead, the expectation maximization (EM) algorithm can be applied [9]. The algorithm starts out with a random initialization of the probabilities, and then iteratively re-estimates the probability of arriving at a local maximum of the likelihood function. It has been shown that the EM algorithm is sensitive to the initialization, and an unlucky initialization results in a non-optimal local maximum. As a solution, clustering of documents has been proposed to initialize the models [16]. Another alternative is latent semantic Dirichlet allocation [15] which has less free parameters, and therefore is less sensitive to the initialization.

Key Applications

This entry focuses on the application of language models to information retrieval. The applications presented include newswire and newspaper search [4,5,15], web search [11], cross-language search [6,19], topic detection and tracking [10,18], and expert search [2]. However, language models have been used in virtually every application that needs processing of natural language texts, including automatic speech recognition, part-of-speech tagging, machine translation, and optical character recognition.

Cross-references

► N-Gram Models
► Probability Smoothing

Recommended Reading

1. Allan J., Aslam J., Belkin N., Buckley C., Callan J., Croft B., Dumais S., Fuhr N., Harman D., Harper D.J., Hiemstra D., Hofmann T., Hovy E., Kraaij W., Lafferty J., Lavrenko V., Lewis D., Liddy L., Manmatha R., McCallum A., Ponte J., Prager J., Radev D., Resnik P., Robertson S., Rosenfeld R, Roukos S., Sanderson M., Schwartz R., Singhal A., Smeaton A., Turtle H., Voorhees E., Weischedel E., Xu J., and Zhai C.X. (eds.). Challenges in information retrieval and language modeling. SIGIR Forum 37(1), 2003.
2. Balog K., Azzopardi L., and Rijke M. Formal models for expert finding in enterprise corpora. In Proc. 29th Annu. Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 43–50.
3. Basharin G.P., Langville A.N., and Naumov V.A. The life and work of A.A. Markov. linear Algebra and its Applications, 386:3–26, 2004.
4. Berger A. and Lafferty J. Information retrieval as statistical translation. In Proc. 22nd ACM Conf. on Research and Development in Information Retrieval, 1999, pp. 222–229.
5. Blei D.M., Ng A.Y., and Jordan M.I. Latent Dirichlet allocation. J. Machine Learn. Res. 3(5):993–1022, 2003.
6. Hiemstra D. and Jong F. Disambiguation strategies for cross-language information retrieval. Lecture Notes in Computer Science, Volume 1696: In Proceedings of the European Conference on Digital Libraries, Springe-Verlag, Berlin Heidelberg New York, 1999, pp. 274–293.
7. Hiemstra D. and Kraaij W. Twenty-One at TREC-7: Ad-hoc and cross-language track. In Proc. 7th Text Retrieval Conference TREC-7. NIST Special Publication 500-242, 1998, pp. 227–238.
8. Hofmann T. Probabilistic latent semantic indexing. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 50–57.
9. Jelinek F. Statistical Methods for Speech Recognition. MIT Press, Cambridge, MA, 1997.
10. Jin H., Schwartz R., Sista S., and Walls F. Topic tracking for radio, TV broadcast and newswire. In Proc. DARPA Broadcast News Workshop, 1999.
11. Kraaij W., Westerveld T., and Hiemstra D. The importance of prior probabilities for entry page search. In Proc. 25th ACM Conf. on Research and Development in Information Retrieval (SIGIR'02), 2002, pp. 27–34.
12. Kraft D.H., Bruce Croft W., Harper D.J., and Zobel J. (eds.). In Proc. 24th ACM Conf. on Research and Development in Information Retrieval. Association for Computing Machinery, 2001.
13. Lavrenko V. and Croft W.B. Relevance models in information retrieval. In Language Modeling for Information Retrieval, W. Bruce Croft and John Lafferty (eds.). Kluwer, Dordecht, 2003, pp. 11–56.
14. Miller D.R.H., Leek T., and Schwartz R.M. A hidden Markov model information retrieval system. In Proc. 22nd ACM Conf. Research and Development in Information Retrieval, 1999, pp. 214–221.
15. Ponte J.M. and Bruce C. W. A language modeling approach to information retrieval. In Proc. 21st ACM Conf. Research and Development in Information Retrieval, 1998, pp. 275–281.
16. Schwartz R.M., Sista S., and Leek T. Unsupervised topic discovery. In Proc. Language Models for Information Retrieval Workshop, 2001.
17. Shannon C.E. A mathematical theory of communication. Bell Syst. Tech. J., 27:379–423, 623–656, 1948.
18. Spitters M. and Kraaij W. Language models for topic tracking. In Language Modeling for Information Retrieval, W. Bruce Croft and J. Lafferty (eds.). Kluwer, Dordecht, 2003, pp. 95–124.
19. Xu J. and Weischedel R. A probabilistic approach to term translation for cross-lingual retrieval. In Language Modeling for Information Retrieval, W. Bruce Croft and John Lafferty (eds.). Kluwer, Dordecht, 2003, pp. 125–140.
20. Zhai C. and Lafferty J. Model-based feedback in the language modeling approach to information retrieval. In Proc. ACM Int. Conf. on Information and Knowledge Management, 2001, pp. 403–410.

Languages for Web Data Extraction

Nicholas Kushmerick
Decho Corporation, Seattle, WA, USA

Synonyms

Web scraping; Screen scraping; Web site wrappers; Web mining; Information extraction

Definition

Web data extraction is the process of automatically converting Web resources into a specific structured format. For example, if a collection of HTML web pages describes details about various companies (name, headquarters, etc) then web data extraction would involve converting this native HTML format into computer-processable data structures, such as entries in relational database tables. The purpose of web data extraction is to make web data available for subsequent manipulation or integration steps. In the previous example, the goal may be summarizing the results as some form of analytical report.

There are several approaches to Web data extraction. The most common approach is to specify the conversion process using a special-purpose programming *Language for Web Data Extraction*. Web data extraction then becomes a matter of executing a well-defined computer program.

Web data extraction languages are generally not intended to be general-purpose programming languages (although some are formally Turing complete). Rather, these special-purpose languages are intended to simplify various actions that are needed to navigate to and retrieve web documents, extract and clean their data, and populate the data into appropriate data structures for output. For example, most languages for web data extraction provide mechanisms for crawling hypertext links, submitting HTML forms, parsing HTML documents, decomposing a document into components using regular expressions, etc. Some languages also offer features such as natural language processing, web-, text- or data-mining algorithms, or access to structured data (e.g., ODBC sources or third-party applications).

This entry focuses primarily on the features of web data extraction languages. However, note that a key strength of some commercial Web data extraction products, and an active area of academic research, is graphical user interface for visually specifying the conversion process (see for example [2]). Similarly, there have been numerous attempts to automatically generate web data extraction programs using machine learning techniques (see for example [3,5,6,9]).

Note also that web data extraction is related to but distinct from web search technologies. Search engines crawl and index large numbers of web documents to facilitate subsequent ad-hoc keyword-based querying. In contrast, web data extractors retrieve a relatively small number of relevant documents (often in a specific sequence as the web sources perform various transactions), and then isolate specific document fragments in order to answer various predefined structured queries on their contents.

Historical Background

As the Web proliferated in the 1990s, computer science researchers from diverse backgrounds (databases, systems, artificial intelligence, information retrieval, etc) realized that the ability to integrate data across heterogeneous sources would give rise to a wide variety of compelling applications, such as shopping assistants that compare products across multiple retail sites. Many researchers who had been investigating traditional formulations of the data integration problem, turned their attention to integrating Web data. This attention revealed many new challenges. Data extraction quickly became prominent among these new challenges. Researchers simply could not demonstrate their Web data integration algorithms in a compelling manner until they developed systematic ways to automatically access large numbers of Web documents and then extract structured data from these documents' native formats.

The first approach to Web data extraction was simply to code up the necessary URL retrieval and data extraction in conventional programming languages. Indeed, many Web data integration applications continue to use this approach today. However, Web data extraction programs implemented in this manner tend to be relatively large, making it difficult to design, debug, re-use and maintain them.

Based on this experience, researchers quickly observed that web data extraction programs written in conventional programs exhibit many common software patterns. This observation led to efforts to encapsulate these patterns either as reusable libraries for existing languages, or as primitives in specialized web data extraction languages (see for example [1,4,7,8,10,11]). For instance, many web data

extraction programs perform crawling behavior such as "parse an HTML document to find all its hyperlinks; then retrieve all newly discovered URLs, and repeat," or "submit a Web form several times, each time binding one of its input parameters to one of several values." To make it easier to build Web crawlers, this behavior could be encoded as a set of functions/classes in a library, or as primitives in a specialized Web data extraction language.

Foundations

From a theoretical perspective, it is meaningless to formally differentiate between Web data extraction languages and "regular" programming languages. Nearly every programming language has features of some aspect of web data extraction, such as retrieving URLs or applying regular expressions. Therefore, nearly any programming language can be used for web data extraction, and so every programming language is a web data extraction language. At the same time, some specialized web data extraction languages are Turing complete, so they can be used to implement arbitrary algorithms that have nothing in particular to do with Web data. Nevertheless, from a practical perspective, it can be much simpler to implement Web data extractors in some languages compared to others. This entry focuses on these practical issues, rather than formal distinctions.

Web data extraction comprises four distinct capabilities:

1. Access: Authenticating to web sites as needed, and fetching particular URLs
2. Extract: Populating data structures from retrieved documents
3. Process: Performing ancillary computations on extracted data
4. Output: Converting data to desired format and delivering to destination

Web data extraction languages vary in the sophistication with which they support these capabilities. All languages support some form of access and extraction, but some provide only limited support for data processing or output.

This entry explains these capabilities by examining one particular Web data extraction language, WebQL. WebQL is a product of QL2 Software Inc (www.QL2.com). While other languages offer a similar range of capabilities, WebQL is a good example for tutorial purposes because it has a familiar SQL-like syntax, and numerous features that address the four capabilities listed above.

WebQL programs consist of one or more *select* statements, which are similar to SQL select statements. When an SQL select statement retrieves values from database tables, a WebQL select statement retrieves data from a Web source. As in SQL, complete WebQL programs comprise a set of select statements that are combined using join, union and other data-flow operations.

Access

The simplest form of Web data access is fetching a single URL. The WebQL program

```
select
   source_content
from
   http://example.com/recipes.html
```

populates a table with one column, and one row, containing the HTML source for the given URL.

As discussed above, it is certainly true that URLs can be easily fetched in every modern programming language. However, web data retrieval languages usually offer a more compact syntax for performing actions that would be much more cumbersome in other languages. For example, the WebQL program

```
select
  source_content
from
  crawl   of   http://example.com/
  recipes.html
  to depth 3
  following if url matching 'potato'
```

implements a spider in WebQL that follows links that satisfy the stated criteria. This compact syntax takes care of numerous low-level details such as timing out, HTTP redirects, HTML parsing, broken links, etc. As a practical matter, this compact syntax generally leads to data extractors that are simpler to design, code, debug, maintain, and reuse. Just as it is formally possible but practically much more difficult to implement complex algorithms in low-level machine-level languages compared to high-level programming languages, web data extractors implemented in special-purpose programming languages are generally simpler to write and easier to extend and maintain.

Another aspect of data access is submitting HTML forms. For example, the WebQL statement

```
select
  source_content
from
  http://example.com/recipe_search
  submitting values ['potato', 'car-
  rot'] for 'ingredient'
```

parses the specified URL to detect HTML forms, and then submits each such form twice, each time binding the input parameter ingredient to one of the specified values. Again, the web data extraction language hides from the programmer details such as binding hidden form parameters.

Extract

After some specific web content has been accessed, web data extractors typically isolate particular fragments or properties of the document. Web programming languages offer numerous capabilities for data extraction. The simplest approach is to use the low-level string operations or regular expression matching capabilities that most programming languages provide. For example, the following WebQL segment extracts two text fragments from a particular URL using low-level operations:

```
select
  -- extract first paragraph, with low-
  level string functions
  substr(instr
  (source_content,'<p>')+3,instr
  (html,'</p>')
  as firstpara,
  -- extract first bold text, with regu-
  lar expression
  extract_pattern
  (source_content,'<b>(.*)</b>')
  as firstbold
from
  http://example.com/recipes.html
```

While these techniques suffice for many simple forms of extraction, most web data extraction languages provide extraction techniques that operate at a higher-level of abstraction. For example, an extractor may need to extract all images and the dimensions from a HTML document. One could do so using a regular expression such as:

```
<img src="([^"]*)" height="([^"]*)"
width="([^"]*)"
```

This is a very brittle approach. It assumes that the attributes are double-quoted, that none are missing, that they occur in the order listed, etc. This regular expression could be refined to accommodate these difficulties, but the resulting very long regular expression would need to be copied each time it is needed.

As an alternative, most web data extraction languages offer a suite of high-level extraction capabilities. For example, one can extract images from HTML documents using WebQL as follows:

```
select
  url,
  height,
  width
from
  images
within
  http://example.com/recipes.html
```

The "from images within" portion of this select statement is an example of a high-level extraction capability. WebQL and other web data extraction languages generally offer a range of such capabilities, including robustly parsing HTML forms, rendering HTML documents into plain text, invoking XPath queries on XML data, using linguistic rules to segment text into paragraphs, sentences, phrases, entities, words, etc, and extracting rows and/or columns from HTML tables.

As an example of the last capability, suppose that http://example.com/recipes.html contains tables embedded with text, such as the following:

Ginger and honey pudding by Nick Nairn Serves 2

Ingredients:		
Amount	**Unit**	**Description**
8	cm	fresh ginger
2		eggs
110	g	butter
110	g	plain flour
110	g	caster sugar
30	g	honey
1	tsp	ground mixed spice
Method:		
1. Grate the ginger into a square of muslin (. . .)		

The following WebQL select statement identifies the tables in this document and then retrieves the values of three specific columns by that are identified by name:

```
select
  amount,
  unit,
  description
from
  table values (amount, unit,
  description)
within
  http://example.com/recipes.html
```

While the details of these extraction techniques vary widely, and while low-level string or regular expression operations could be used to extract this data in any modern programming language, the general point holds: by providing abstract high-level data extraction operations, web data extraction languages make it much easier to implement extractors that are concise, correct, maintainable, and reusable.

Process

So far, this entry has focused on the advanced access and extraction functionality of web data extraction languages, compared to traditional programming languages. However, real web data extractors usually need to do more than simply access documents and extract their data; they also need to process this data in some way.

Web data extraction languages generally have a rich set of traditional programming language operators and control structures for manipulating and transforming data. For example, WebQL offers most of the dataflow operations in SQL dialects, such as joins, unions, grouping, sorting, and removing duplicates. WebQL also has a wide variety of functions for performing many kinds of operations. To illustrate these capabilities, consider the following query:

```
select as ExistingMeat
  ingredient as existing_ingredient,
  text_to_datetime("best before",
  'dd-mm-yy')
  as best_before_date
from
  table values (ingredient, category, "best before")
within
  http://www.example.com/current_fridge_contents.html
where
  category = 'meat'
join to ExistingMeat
select as RequiredIngredient
  description as
  required_ingredient,
  existing_ingredient
from
  table values (description)
within
  http://example.com/recipe_search
  submitting values existing_ingredient for 'ingredient'
join to ExistingMeat,
RequiredIngredient
  where ExistingMeat.existing_ingredient =
  RequiredIngredient.
  existing_ingredient
select
  required_ingredient,
  best_before_date < now() + 24*60*60
  as urgent
sort by
  best_before_date desc
```

For the sake of this entry, there is no need to explain this program in detail. At a high level, it first extracts a list of ingredients currently in the refrigerator along with their category (meat, vegetables, etc) and expiration date; then requests recipes containing each meat that is in stock, and records which other ingredients are needed to make those recipe. And finally the program sorts the required ingredients by the expiration date of their meat, and also outputs a Boolean value that is true if the meat will soon expire. This example illustrates the ability to join intermediate tables, sort data, perform numerical and logical operations, and transform textual date/time values into a native numerical representation.

Output

After accessing, extracting and processing the required data, web data extractors must deliver the data to some destination in the appropriate format. Web data extraction languages vary in the range of output

formats and delivery mechanisms that they support. To illustrate some of these capabilities, consider the following WebQL program:

```
select
  url,
  content
from
  links
within
  http://www.example.com/recipes.
  html
into
  -- write XML to local file
  file:links.xml,
  -- send CSV to an FTP site
  ftp://jsmith@mysecret:ftp.exam-
  ple.com/links.csv,
  -- write data as new rows in a database
  table
  'links'@docdb
```

The "into" clause causes the data to be encoded in specific formats and sent to specific destinations. This example program extracts the links from a web page, and then delivers them in various formats (comma-separated text file, XML) to both a local file and an FTP site, and also inserts the data directly into database table via ODBC.

Key Applications

Web data extraction languages are widely used in numerous commercial applications across a wide variety of industries The most common scenario is that an enterprise wants to extract data from external Web sources for which public Web Services or other programmatic APIs are not available. While numerous (often "Web 2.0"-oriented) data sources offer public APIs or structured data feeds such as RSS, many more sources do not. This is particularly true for high-value data extraction applications such as the gathering of real-time competitive market intelligence. For example, most large retailers, manufacturers and distributors employ some form of Web data extraction to monitor their competitor's prices. In most cases, the only way to harvest such data is by extracting it from public Web sources such as on-line retail catalogs or shopping sites.

Web data extraction is also used as a form of lightweight data integration for interconnecting legacy applications within an enterprise. Many companies are not in a position to modify critical business systems used for inventory management, pricing, customer management, etc., yet they want to integrate these systems. Web data extraction tools are frequently used to facilitate such integration by working with existing Web-based interfaces, rather than forcing the enterprise to modify their legacy systems.

A third important application area is so-called "mashups," applications that combine data from multiple sources into unified service. Web data extraction languages are clearly an appropriate technology with which to develop mashups. To many proponents, a key aspect of the mashup philosophy is the ability of ordinary users to generate their own applications. Therefore, most mashup technologies focus on visual interfaces that allows non-technical users to develop web data extraction programs.

While this entry focuses largely on technical issues, it is important to point out that the use of automated web data extractors may give rise to two sorts of legal issues. First, the terms of use of many Web sites explicitly prohibit automated content extraction. Second, republication of the extracted data may violate the original source's copyright on the data.

Future Directions

The core theoretical and scientific issues regarding web data extraction are well understood. While numerous companies are competing in the web data extraction/integration space, they compete primarily on the basis of the value-added services they deliver on top of the extracted data, rather than on the basis of their extraction technology. Nevertheless, as web technologies change, web data extraction languages will be enhanced to accommodate formats such as Flash/AMF in which more and more web data is embedded.

More significantly, some "web" data extraction languages provide access to a wide range of non-web sources (such as email, FTP sites, execution of external programs, etc) and formats (such as office documents and spreadsheets, archived and compressed data, etc). As these capabilities expand, these languages will not remain focused exclusively on web data, but rather they will evolve into powerful general-purpose tools for manipulating and integrating arbitrary structured and unstructured digital content.

URL to Code

There are several open-source Web data extraction tools, usually in the form of a library for a full-fledged programming language. Examples include:

1. Perl: WWW::Mechanize (search.cpan.org/dist/WWW-Mechanize)
2. PHP: Curl (www.php.net/curl)
3. Java: Web-Harvest (web-harvest.sourceforge.net)

Many companies compete in the Web data extraction market. These companies generally offer standalone data-extraction products, as well as on-demand services based on these technologies that are tailored to specific vertical industries. Examples include:

1. QL2 (www.ql2.com)
2. Fetch (www.fetch.com)
3. Kapow (www.kapowtech.com)
4. Lixto (www.lixto.com)
5. Connotate (www.connotate.com)

Cross-references

- ▶ Content-and-Structure Query
- ▶ Data Cleaning
- ▶ Data Integration
- ▶ Data Integration in Web Data Extraction Systems
- ▶ Database Reverse Engineering
- ▶ Focused Web Crawling
- ▶ Fully Automatic Web Data Extraction
- ▶ Hidden-Web Search
- ▶ Incremental Web Crawling
- ▶ Indexing Semi-Structured Data
- ▶ Indexing the "Web"
- ▶ Information Extraction
- ▶ Information Integration
- ▶ Logical Foundations for Web Data Extraction
- ▶ Metasearch Engines
- ▶ Query Language
- ▶ Screen Scraper
- ▶ Semantic Web
- ▶ Semi-Structured Data
- ▶ Semi-Structured Query Language
- ▶ Semi-Structured Text
- ▶ Structured Document Retrieval
- ▶ Text Mining
- ▶ Text Normalization
- ▶ Web Crawler Architecture
- ▶ Web Data Extraction System
- ▶ Web Harvesting
- ▶ Web Information Extraction
- ▶ Wrapper Induction
- ▶ Wrapper Maintenance
- ▶ Wrapper Stability
- ▶ XML Information Integration
- ▶ Xpath/XQuery

Recommended Reading

1. Arasu A. and Garcia-Molina H. Extracting structured data from Web pages. In Proc. 2003 ACM SIGMOD Int. Conf. on Management of data, 2003, pp. 337–348.
2. Baumgartner R., Flesca S., and Gottlob G. Visual Web information extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 119–128.
3. Crescenzi V., Mecca G., and Merialdo P. RoadRunner: towards automatic data extraction from large web sites. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
4. Kistler T. and Marais H. WebL – a programming language for the Web. Comput. Netw. ISDN Syst., 30(1–7):259–270, 1998.
5. Knoblock CA., Lerman K., Minton S., and Muslea I. Accurately and reliably extracting data from the web: A machine learning approach. In Intelligent Exploration of the Web, Piotr S. Szczepaniak, Javier Segovia, Janusz Kacprzyk and Lotfi A. Zadeh (eds.). Physica-Verlag Heidelberg, Heidelberg, pp. 275–287, 2003.
6. Kushmerick N. Wrapper induction: efficiency and expressiveness. Artif. Intell., 118(1–2):15–68, 2000.
7. Laender A.H.F., Ribeiro-Neto B.A., da Silva A.S., and Teixeira J.S. A brief survey of web data extraction tools. ACM SIGMOD Record, 31(2):84–93, 2002.
8. Liu L., Pu C., and Han W. XWRAP: an XML-enabled Wrapper Construction System for Web Information Sources. In Proc. 16th Int. Conf. on Data Engineering, 2000.
9. Muslea I., Minton S., and Knoblock C.A. Hierarchical wrapper induction for semistructured information sources. J. Auton. Agents Multi-Agent Syst., 4(1–2):93–114, 2001.
10. Sahuguet A. and Azavant F. Building intelligent web applications using lightweight wrappers. Data and Knowledge Engineering, 36(3):283–316, 2000.
11. Spertus E. and Andrea Stein L. Squeal: structured queries on the Web. In Proc. 9th Int. World-Wide Web Conference, 2000.

Large Itemsets

- ▶ Frequent Itemsets and Association Rules

Latch Coupling

- ▶ B-Tree Locking

Latching

▶ B-Tree Locking

Latent Semantic Indexing

▶ Singular Value Decomposition

Layer Algebra

▶ Spatial Operations and Map Operations

Layered Architecture

▶ Temporal Strata

Layered Transactions

▶ Multilevel Transactions and Object-Model Transactions

Lazy Replication

▶ Optimistic Replication and Resolution

LBS

▶ Location-based Services

Lévy Skew a-Stable Distribution

▶ Stable Distributions

Learning Distance Measures

Carlotta Domeniconi
George Mason University, Fairfax, VA, USA

Synonyms

Flexible metric computation; Adaptive metric techniques

Definition

Many problems in data mining (e.g., classification, clustering, information retrieval) are concerned with the discovery of homogeneous groups of data according to a certain similarity (or distance) measure. The distance measure in use strongly affects the nature of the patterns (clusters, classes, or retrieved images) emerging from the given data. Typically, any chosen fixed distance measure, such as Euclidean or Manhattan distance, does not capture the underlying structure of the data, and fails to find meaningful patterns which correspond to the user's preferences. To address this issue, techniques have been developed that learn from the data how to compute dissimilarities between pairs of objects. Since objects are commonly represented as vectors of measurements in a given feature space, distances between two objects are computed in terms of the dissimilarity between their corresponding feature components. In this setting, learning a (local) distance measure for a query object $\mathbf{q} \in \Re^n$ (to be classified) and an arbitrary object $\mathbf{x} \in \Re^n$ means to learn a weighted p-norm distance metric on the Euclidean space of the input measurement variables (or features):

$$D_p(\mathbf{q}, \mathbf{x}) = \{\sum_{i=1}^{n} |[W(\mathbf{q})(\mathbf{q} - \mathbf{x})]_i|^p\}^{1/p},$$

where $p > 0$ and $W(\mathbf{q}) \in \Re^{n \times n}$ is a matrix of weights reflecting the relevance or importance of features at the query $\mathbf{q}$. If $W(\mathbf{q})$ depends on the query point $\mathbf{q}$, the resulting distance measure is *local*; otherwise, if W is invariant to the query, a *global* distance measure is obtained.

Historical Background

The problem of learning distance measures from data has attracted considerable interest recently in the data mining and machine learning communities. Different methodologies have been developed for supervised, unsupervised, and semi-supervised problems. One of the earliest work that discusses the problem of clustering simultaneously both points and features is Hartigan, 1972 [9]. A model based on direct clustering of the data matrix and a distance-based model are introduced, both leading to similar results.

Foundations

The ability to classify patterns is certainly one of the key features of intelligent behavior, whether it is

humans' or animals'. This ability emerged with the biogenetic evolution for survival purposes, not only of individuals but also of entire species. An individual receives sensory information that must be processed to perceive, and ultimately act, possibly for orientation in the environment, distinction between edible and poisonous food, or detection of dangerous enemies.

Machine perception and classification as features of artificial systems serve similar but more constrained purposes. Machine classification aims to provide artificial systems with the ability to react to situations and signals that come from the environment to perform specific tasks. As such, pattern classification is a fundamental building block of any cognitive automata.

Recent developments in data mining have posed new challenges to pattern classification. Data mining is a knowledge discovery process whose aim is to discover unknown relationships and/or patterns from a large set of data that make it is possible to predict future outcomes. As such, pattern classification becomes one of the key steps in attempting to uncover the *hidden knowledge* within the data. The primary goal is usually predictive accuracy, with secondary goals being speed, ease of use, and interpretability of the resulting predictive model.

The term *pattern* is a common word and means something exhibiting some form of regularity, able to serve as a model representing a concept of what was observed. As a consequence, a pattern is never an isolated observation, but rather a collection of observations connected in time or space (or both). A pattern exhibits, as a whole, a certain structure indicative of the underlying concept. The pattern classification task can then be seen as the task of inferring concepts from observations. Thus, designing a pattern classifier means defining a mapping from a measurement space into the space of possible meanings, that are viewed as finite and discrete target points.

From this perspective, it makes no difference what kind of observations are considered and to what kind of meanings they may be linked. The same approach can be used to recognize written text, spoken language, objects, or any other multidimensional signals as well. The selection of meaningful observations from a specific domain is a *feature extraction* process. From a theoretical viewpoint, the distinction between feature extraction and classification is arbitrary, but nevertheless useful. In general, the problem of feature extraction is much more domain dependent than the problem of classification.

Statistical Approach

A characteristic of patterns in the context of classification is that every concept (or class) may have multiple representative points in the measurement space. For example, for the task of character recognition from their images, there exists a potentially unlimited plurality of ways to design character images that correspond to the same character. Therefore, the very core of pattern classification is to cope with variability. The difficulty of the task depends on the degree to which the representatives of a class are allowed to vary and how they are distributed in the measurement space. This observation brings together two intrinsic components of the pattern classification task: the *statistical* component and the principle of *learning from examples*.

The problem of classification can be seen as one of partitioning the feature space into regions, one region for each category. Ideally, one would like to arrange this partitioning so that no decisions is ever wrong. This objective may not be achievable for two reasons. The distributions of points of different classes in the measurement space overlap; thus, it is not possible to reliably separate one class from the other. Moreover, even if a rule that does a good job of separating the examples can be found, one has no guarantee that it will perform as well on new points. In other words, that rule may not *generalize* well on data never seen before. It would certainly be safer to consider more points, and check how many of those are correctly classified by the rule. This suggests that one should look for a classification procedure that aims at minimizing the *probability of error*. The problem of classification then becomes a problem in statistical decision theory.

Challenges

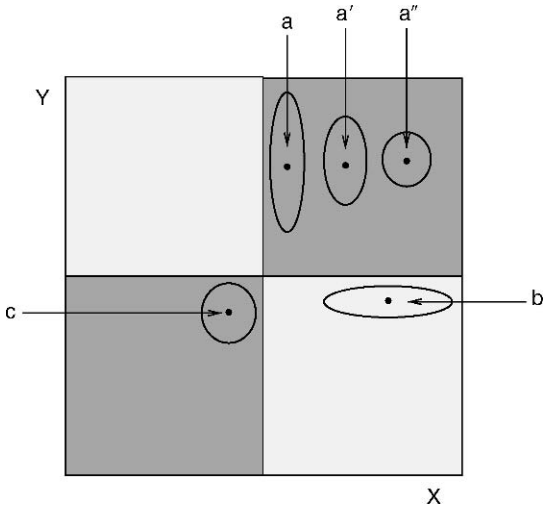
While pattern classification has shown promise in many areas of practical significance, it faces difficult challenges from real world problems, of which the most pronounced is Bellman's *curse of dimensionality* [1]. It states the fact that the sample size required to perform accurate prediction in problems with high dimensionality is beyond feasibility. This is because in high dimensional spaces, data become extremely sparse and are apart from each other. As a result, severe bias that affects any estimation process can be introduced in a high dimensional feature space with finite samples.

Consider, for example, the rule that classifies a new data point with the label of its closest training point in the measurement space (*1-Nearest Neighbor rule*). Suppose each instance is described by 20 attributes, but only three of them are relevant to classifying a given instance. In this case, two points that have identical values for the three relevant attributes may nevertheless be distant from one another in the 20-dimensional input space. As a result, the similarity metric that uses all 20 attributes will be misleading, since the distance between neighbors will be dominated by the large number of irrelevant features. This shows the effect of the curse of dimensionality phenomenon, that is, in high dimensional spaces distances between points within the same class or between different classes may be similar. This fact leads to highly biased estimates. Nearest neighbor approaches are especially sensitive to this problem.

In many practical applications things are often further complicated. In the previous example, the three relevant attributes for the classification task at hand may be dependent on the location of the query point, i.e., the point to be classified, in the feature space. Some features may be relevant within a specific region, while other features may be more relevant in a different region.

These observations have two important implications. Distance computation does not vary with equal strength or in the same proportion in all directions in the feature space emanating from the input query. Moreover, the value of such strength for a specific feature may vary from location to location in the feature space. Figure 1 illustrates a case in point, where class boundaries are parallel to the coordinate axes. For query *a*, dimension *X* is more relevant, because a slight move along the *X* axis may change the class label, while for query *b*, dimension *Y* is more relevant. For query *c*, however, both dimensions are equally relevant. Capturing such information, therefore, is of great importance to any classification procedure in high dimensional settings.

It is important to emphasize that the curse of dimensionality is not confined to classification. It affects any estimation process in a high dimensional feature space with finite examples. Thus, clustering equally suffers from the same problem. The clustering problem concerns the discovery of homogeneous groups of data according to a certain similarity measure. It is not meaningful to look for clusters in high dimensional spaces as the average density of points anywhere in input space is likely to be low. As a consequence, distance functions that equally use all input features may not be effective.



Learning Distance Measures. Figure 1. Feature relevance varies with query locations.

Adaptive Metric Techniques

This section presents an overview of relevant work in the literature on flexible metric computation for classification and clustering problems.

Adaptive Metric Nearest Neighbor Classification

In a classification problem, one is given l observations $\mathbf{x} \in \Re^n$, each coupled with the corresponding class label y, with $y = 1,...,J$. It is assumed that there exists an unknown probability distribution $P(\mathbf{x}, y)$ from which data are drawn. To predict the class label of a given query $\mathbf{q}$, the class posterior probabilities $\{P(j|\mathbf{q})\}_{j=1}^{J}$ need to be estimated.

K nearest neighbor methods are based on the assumption of smoothness of the target functions, which translates to locally constant class posterior probabilities $P(j|\mathbf{q})$, that is: $P(j|(\mathbf{q} + \delta\mathbf{q})) \simeq P(j|\mathbf{q})$, for $\|\delta\mathbf{q}\|$ small enough. Then, $P(j|\mathbf{q}) \simeq \frac{\sum_{\mathbf{x}\in N(\mathbf{q})} P(j|\mathbf{x})}{|N(\mathbf{q})|}$, where $N(\mathbf{q})$ is a neighborhood of $\mathbf{q}$ that contains points $\mathbf{x}$ that are "close" to $\mathbf{q}$, and $|N(\mathbf{q})|$ denotes the number of points in $N(\mathbf{q})$. This motivates the estimates

$$\hat{P}(j|\mathbf{q}) = \frac{\sum_{i=1}^{l} 1(\mathbf{x}_i \in N(\mathbf{q}))1(y_i = j)}{\sum_{i=1}^{l} 1(\mathbf{x}_i \in N(\mathbf{q}))},$$

where $1()$ is an indicator function such that it returns 1 when its argument is true, and 0 otherwise.

The assumption of smoothness, however, becomes invalid for any fixed distance metric when the input observation approaches class boundaries. The objective of locally adaptive metric techniques for nearest neighbor classification is then to produce a modified local neighborhood in which the posterior probabilities are approximately constant.

The techniques proposed in the literature [5,6,10] are based on different principles and assumptions for the purpose of estimating feature relevance locally at query points, and therefore weighting accordingly distances in input space. The idea common to these techniques is that the weight assigned to a feature, locally at a given query point **q**, reflects its estimated relevance to predict the class label of **q**: larger weights correspond to larger capabilities in predicting class posterior probabilities. As a result, neighborhoods get constricted along the most relevant dimensions and elongated along the less important ones. The class conditional probabilities tend to be constant in the resulting neighborhoods, whereby better classification performance can be obtained.

Large Margin Nearest Neighbor Classifiers

The previously discussed techniques have been proposed to try to minimize bias in high dimensions by using locally adaptive mechanisms. The "lazy learning" approach used by these methods, while appealing in many ways, requires a considerable amount of on-line computation, which makes it difficult for such techniques to scale up to large data sets. Recently, a method (called LaMaNNA) has been proposed which, although still founded on a query based weighting mechanism, computes off-line the information relevant to define local weights [3].

The technique uses support vector machines (SVMs) as a guidance for the process of defining a local flexible metric. SVMs have been successfully used as a classification tool in a variety of areas [13], and the maximum margin boundary they provide has been proved to be optimal in a structural risk minimization sense. The decision function constructed by SVMs is used in LaMaNNA to determine the most discriminant direction in a neighborhood around the query. Such direction provides a local feature weighting scheme. This process produces highly stretched neighborhoods along boundary directions when the query is close to the boundary. As a result, the class conditional probabilities tend to be constant in the modified neighborhood, whereby better classification performance can be achieved. The amount of elongation-constriction decays as the query moves farther from the vicinity of the decision boundary. This phenomenon is exemplified in Fig. 1 by queries a, a' and a''.

Adaptive Metrics for Clustering and Semi-Supervised Clustering

Adaptive metric techniques for data without labels (unsupervised) have also been developed. Typically, these methods perform clustering and feature weighting simultaneously in an unsupervised manner [2,4,7,8,12]. Weights are assigned to features either globally or locally.

The problem of feature weighting in K-means [11] clustering has been addressed in [12]. Each data point is represented as a collection of vectors, with "homogeneous" features within each measurement space. The objective is to determine one (global) weight value for each feature space. The optimality criterion pursued is the minimization of the (Fisher) ratio between the average within-cluster distortion and the average between-cluster distortion.

COSA (Clustering On Subsets of Attributes) [7] is an iterative algorithm that assigns a weight vector (with a component for each dimension) to each data point. COSA starts by assigning equal weight values to each dimension and to all points. It then considers the k nearest neighbors of each point, and uses the resulting neighborhoods to compute the dimension weights. Larger weights are credited to those dimensions that have a smaller dispersion within the neighborhood. These weights are then used to compute dimension weights for each pair of points, which in turn are utilized to update the distances for the computation of the k nearest neighbors. The process is iterated until the weight values become stable. At each iteration, the neighborhood of each point becomes increasingly populated with data from the same cluster. The final output is a pairwise distance matrix based on a weighted inverse exponential distance that can be used as input to any distance-based clustering method (e.g., hierarchical clustering).

LAC (Locally Adaptive Clustering) [4] develops an exponential weighting scheme, and assigns a weight vector to each cluster, rather than to each data point. The weights reflect local correlations of data within each discovered cluster, and reshape each cluster as a dense spherical cloud. The directional local reshaping

of distances better separates clusters, and allows for the discovery of different patterns in different subspaces of the original input space.

Recently, to aid the process of clustering data according to the user's preferences, a semi-supervised framework has been introduced. In this scenario, the user provides examples of similar and dissimilar points, and a distance metric is learned over the input space that satisfies the constraints provided by the user [14].

Key Applications

Almost all problems of practical interest are high dimensional. Thus, techniques that learn distance measures have significant impact in fields and applications as diverse as bioinformatics, security and intrusion detection, document and image retrieval. An excellent example, driven by recent technology trends, is the analysis of microarray data. Here one has to face the problem of dealing with more dimensions (genes) than data points (samples). Biologists want to find "marker genes" that are differentially expressed in a particular set of conditions. Thus, methods that simultaneously cluster genes and samples are required to find distinctive "checkerboard" patterns in matrices of gene expression data. In cancer data, these checkerboards correspond to genes that are up- or downregulated in patients with particular types of tumors.

Cross-references

- ▶ Classification
- ▶ Cluster and Distance Measure
- ▶ Clustering with Constraints
- ▶ Curse of Dimensionality
- ▶ Data Mining
- ▶ Feature Selection for Clustering
- ▶ Nearest Neighbor Classification

Recommended Reading

1. Bellman R. Adaptive Control Processes. Princeton University Press, 1961.
2. Blansch A., Ganarski P., and Korczak J. Maclaw: a modular approach for clustering with local attribute weighting. Pattern Recognit. Lett., 27(11):1299–1306, 2006.
3. Domeniconi C., Gunopulos D., and Peng J. Large margin nearest neighbor classifiers. IEEE Trans. Neural Netw., 16:899–909, 2005.
4. Domeniconi C., Gunopulos D., Yan S., Ma B., Al-Razgan M., and Papadopoulos D. Locally adaptive metrics for clustering high dimensional data. Data Mining Knowl. Discov. J., 14:63–97, 2007.
5. Domeniconi C., Peng J., and Gunopulos D. Locally adaptive metric nearest neighbor classification. IEEE Trans. Pattern Anal. Mach. Intell., 24:1281–1285, 2002.
6. Friedman J. Flexible metric nearest neighbor classification. In Tech. Report, Dept. of Statistics, Stanford University, 1994.
7. Friedman J. and Meulman J. Clustering Objects On Subsets of Attributes. Technical Report, Stanford University, 2002.
8. Frigui H. and Nasraoui O. Unsupervised learning of prototypes and attribute weights. Pattern Recognit., 37(3):943–952, 2004.
9. Hartigan J.A. Direct clustering of a data matrix. J. Am. Stat. Assoc., 67(337):123–129, 1972.
10. Hastie T. and Tibshirani R. Discriminant adaptive nearest neighbor classification. IEEE Trans. Pattern Anal. Machine Intell., 18:607–615, 1996.
11. Jain A., Mutty M., and Flyn P. Data clustering: a review. ACM Comput. Surv., 31(3), 1999.
12. Modha D. and Spangler S. Feature weighting in K-means clustering. Mach. Learn., 52(3):217–237, 2003.
13. Shawe-Taylor J. and Fiege N. Pietzuch P. Kernel Methods for Pattern Analysis. Cambridge University Press, London, 2004.
14. Xing E., Ng A., Jordan M., and Russell S. Distance metric learning, with application to clustering with side-information. Advances in NIPS, vol. 15, 2003.

Learning in Streams

▶ Classification on Streams

Length Normalization

▶ Document Length Normalization

Level-of-Detail (LOD) Terrain Modeling

▶ Multi-Resolution Terrain Modeling

Levelwise Search

▶ Apriori Property and Breadth-First Search Algorithms

Lexical Affinities

▶ Term Proximity

Lexical Analysis of Textual Data

Chris D. Paice
Lancaster University, Lancaster, UK

Synonyms

Lexical processing; Term processing

Definitions

Lexical analysis refers to the association of meaning with explicitly specified textual strings, referred to here as *lexical terms*. These lexical terms are typically obtained from texts (whether natural or artificial) by a process called *term extraction*. The association of meaning with lexical terms involves a data structure known generically as a *lexicon*. The characteristic operation in using a lexicon is a *lookup*, where the input is a lexical term, and the output is a representation of one or more associated meanings. A lexicon consists of a collection of *entries*, each of which comprises an *entry term* and a meaning structure. Lookup entails finding any entries whose entry term matches the lexical term in question.

Here, the term lexical analysis is used to refer only to operations performed on complete words or word groups. Operations on the characters within words is the concern of *morphology*.

The use of *text corpora* for obtaining data on the properties of words may also be regarded as a branch of lexical analysis. A text corpus provides a large representative sample of a language or sublanguage, and may be used for generating data-sets such as lexicons (see later), for providing statistical information, and for identifying examples of particular language constructs for researchers. Operations on corpora include annotating words or word groups with grammatical or other information. [9,11]

Historical Background

The need for automatic lexical analysis dates back to the 1950s, when the need arose for programing language compilers to recognize variable names and reserved words, and assign an appropriate role to each one. As soon as computers started being used for processing natural language texts, lexical processing was required for extracting, recognizing, organizing, and correlating words and phrases. At first, lexicons for term recognition were compiled by hand, or adapted from existing dictionaries and lists, but later, automatic tools were developed, either for generating lexicons outright, or for reducing the amount of human effort involved in constructing them.

Foundations

A natural language word is a sign whose meaning cannot be inferred (except in special cases such as onomatopoeic words) from its phonological or morphological structure. *Lexical analysis* is a process by which meanings are associated with specific words or other textual strings. These strings, which can be referred to as *lexical terms*, or just *terms*, are typically extracted during the scanning of some document.

Lexical terms may be individual words belonging to a natural or artificial language, but they may also include abbreviations such as "IBM" and "USAAF", 'pseudo-words' such as "PL/1" and "B12", and even multiword expressions (MWEs) such as "winter wheat" or "Boeing 747." The key point is that the term must represent a known concept which is, or may be, relevant to some current need. The function of lexical analysis is to return information enabling that need to be satisfied.

A data structure or data collection which associates a set of lexical terms with their meanings is known as a *lexicon*. The meaning associated with a specified lexical term may be referred to as an *output* of the lexicon. The structure of the lexicon and the nature of the output vary according to what kind of information is required for the task at hand. Some applications may require access to two or more different lexicons, covering different (or partly different) sets of terms, and yielding different kinds of output.

Although the detailed arrangements can vary, a lexicon consists essentially of a set of *entries*, each of which maps a term to a meaning. In some lexicons, a given term can yield a number of alternative meanings. In such a case, the application program must choose the most appropriate meaning or (depending on the task at hand) make use of them all.

The archetypal lexicon is a traditional *dictionary*, in which each natural language word is accompanied by a definition of each distinct meaning of the word, together with indicators for pronunciation, syntax, etymology, etc. [11]

In a lexicon, the key operation is the *lookup*, which takes a term and returns an output representing the required meaning or meanings. In cases where the term is not present in the lexicon, a "null" or "false" output is returned.

In applications where the range of terms and meanings are very small, lexical analysis may be performed by encoding (or "hard wiring") the relevant terms as literals in decision structures in a computer program.

Lexical Analysis in Text Processing

1. Lexical extraction Reference to a lexicon is frequently required by an application program designed to process textual data, whether this consists of text in a natural language, or in an artificial language such as a programming language. In extracting terms from a text, the first step is normally to *tokenize* the text – that is, to divide the stream of characters into coherent groups called *tokens*, as appropriate for the task at hand. In a natural text, the tokens may include words, numbers, punctuation symbols, brackets, etc.

As noted earlier, lexical analysis typically involves the lookup of significant items such as words and phrases. Tokenization therefore usually involves, or is immediately followed by, a filtering process which recognizes and discards irrelevant items such as punctuation marks. In fact, in a language like English, extraction of single words can be achieved by regarding all characters except letters, and perhaps digits, as token delimiters. There remain only certain specific issues, such as the handling of hyphens and apostrophes, where the extracted terms need to conform to the practice used in the lexicon. [6,8]

If the lexicon includes terms which are multiword expressions (MWEs), there is a problem in that MWEs are not explicitly flagged in a text, and so cannot be extracted directly. The only exception is the names of places, people and organizations, such as "Department for Transport" or "United Arab Emirates," where the main component words are capitalized.

Lexical terms are almost always represented by grammatical noun phrases (NPs). Hence, one approach to the extraction of MWEs is to parse the text and then extract the NPs as potential terms. Parsing is however a slow process, and only likely to be worthwhile if the output of the parser is to be used for other purposes as well. Fortunately, NPs can also be identified by a more limited syntactic process based on recognizing patterns of grammatical tags, which can be done reasonably quickly. [11]

A more simple-minded approach to term extraction involves generating all sequences of successive tokens up to a specified length, and checking each against the lexicon in decreasing order of length. With a four-token limit, this would seem to involve checking four potential terms per token, but in fact this is an overestimate. Firstly, sequences starting or finishing with a trivial word like "of" or "for", or containing a punctuation symbol, might be discarded at once (though this begs the question of how efficiently these cases can be recognized). Secondly, if the first word of a long sequence fails to match, the shorter sequences starting with that word need not be tested. All in all, with its avoidance of syntactic processing, this can be quite an effective approach.

A different method is to only extract single tokens. but if the current token happens to match the first word of a multiword term, the succeeding tokens of the text are checked against the remainder of that entry (and also against any other entries starting with the same word).

If it is known that the terms of interest will only occur in certain specific contexts, rules can be used for defining those contexts. For instance, a rule may be devised for extracting surnames from personal names, covering examples such as "Mrs Robinson," "John F. Ratley, Jr," and "Revd Ben Wiles". Alternatively, data may need to be extracted from semiformatted texts such as clinical reports. Rules of this kind may consist of, or be based on, regular expressions – see URL below, and reference [4].

As noted later, extracted terms are sometimes compiled into an index before further processing, but for the present it is assumed that each term is immediately passed on for lookup.

Besides terms extracted from a text, lexical processing may need to be applied to terms retrieved from a database or knowledge base, extracted from a user query, or entered into a text box in a user interface. In some cases, the lexical terms might be artificial values, such as employee numbers assigned by a company or internal identifiers assigned by a database system.

2. *Lookup* For a lexicon of any size, the efficiency of the lookup is of major importance. The techniques here are well known, including simple binary searching in arrays, the use of access trees, and hashing techniques. For fixed sets of terms, hashing is potentially the fastest of all methods, but if terms are likely to be added or deleted frequently access trees are probably better. Further details about these methods can be found in any good book on data structures.

Of course, in the case of a lexicon designed for use by human beings, alphabetical ordering is the almost invariable rule.

3. *Use of the Output* The output of a lexicon, unless it is simply to be displayed on a user's workstation, will be subjected to further processing by the application program. As noted earlier, the output of each lexicon is designed according to its intended uses.

Each entry in a lexicon consists of an entry term and an output structure. Besides the outputs explicitly defined in the lexicon, there is normally a default output which is returned in cases where the lexical term is not included among the entry terms.

The simplest form of lexicon is one in which the explicit outputs are all void; in other words, the lexicon is simply a *list* of terms. In effect, the lookup process returns the value "true" or "false," depending on whether or not the term was found. Such a lexicon defines a *set* of terms, and the lookup is a test for *set membership*.

The simplest form of nonvoid lexicon is a simple association list, in which each output is a single value. In general, however, the outputs can have an arbitrarily complicated structure, as defined by two factors: multiplicity and complexity.

Multiplicity of output refers to the fact that, in many lexicons, a given entry term may be associated with a number of different outputs (or equivalently, there may be a number of separate entries which have the same entry term). In many natural language tasks, multiplicity of output equates to ambiguity (as in the various distinct meanings of the English word "dock"), which represent a problem to be resolved. In other cases (e.g., when the outputs enumerate the members of a set) the multiplicity is not a problem; for example, in an inverted file (see below) multiple outputs are the norm.

Complexity of output. In many lexicons each output consists of several differentiated data items. The output may for example be structured as a record or tuple, or it may have a deeper structure, as in some examples below.

Types of Lexicon

Word lists and Stoplists As noted above, the simplest form of lexicon is a simple term list representing a set of terms which share some common property. Membership of such a set can result in the acceptance of a term for further processing by the application, or alternatively can cause its rejection. An example of the latter case is a *stoplist*, which is routinely used in information retrieval systems to eliminate trivial words ("stopwords") which can serve no purpose for retrieval. These are mainly syntactic function words ("closed class words") such as the English words "the," "of," "and," "for," "over," "because," etc.

Indexes and Inverted files An *index* is a secondary structure which is provided to facilitate access to the items held in some primary body of information. A back-of-book index provides a familiar everyday example, but computer-based indexes can be generated automatically for any kind of information object which includes lexical data. An important example is the use of *inverted files* for assisting the retrieval of records in databases. Given a database in which each record contains a set of terms summarizing the properties of a particular entity, inspecting all of the records individually will likely be a slow process. Instead, an inverted file is generated in which each distinct term is associated with an entry listing the identifiers of the specific records which contain that term. Searching can then be performed quickly by accessing and comparing only those few entries which relate to the specified search terms [6].

Machine Readable Dictionaries The outputs of traditional dictionaries and glossaries were designed for perusal by human beings. However, with their generally predictable structure, they proved to be a valuable lexical resource for natural language processing applications [2]. Their great advantage was that they provided a ready-made summary of the whole of the language, apart from specialized technical terms. Their disadvantage was that their presentation of information was often inconsistent and incomplete. Experience of such problems soon led to the development of electronic dictionaries with more formally defined structures; perhaps the best-known example is the Longman Dictionary of Contemporary English (LDOCE), which is now available on CD-ROM.

Thesauruses A thesaurus is a structure which records and classifies relationships between distinct lexical terms. Probably its best-known use is for detection of synonyms, to enable distinct but equivalent terms to be merged or conflated. In information retrieval, a thesaurus may be used for expanding a set of query terms, thus improving the recall performance of a query. Some

of the most detailed thesauruses provide information about the terminology of a technical domain, such as medicine or agriculture.

In fact, a well-developed thesaurus provides information about a range of distinct interterm relationships, and may support two distinct types of operation: (i) vocabulary control, and (ii) classification of relationships. Vocabulary control consists of mapping each lexical term onto a *preferred term*, denoting some specific concept in the domain, thus effectively conflating different (synonymous) mentions of that concept. Such a thesaurus also includes information about semantic relationships between distinct concepts, including antonymy and various hierarchical relations. The latter almost always include genus/type relations (e.g., plant/tree/oak), and often also whole/part relations (e.g., tree/branch/twig) and others. [11]

The WordNet is an online thesaurus developed at Princeton University in the early 1990s [5,10]. In WordNet, groups of synonymous words are organized into "synsets" (but no preferred terms are designated). Words which are polysemous (i.e., possess more than one meaning) will occur in two or more synsets – for example, (board, plank) and (board, committee). This means that the entry term "board" returns two distinct outputs. WordNet also allows five further relations to be recorded between synsets – antonymy, genus/type, whole/part, manner, and logical entailment.

Although WordNet is an attractive and useful tool, it is really quite "weak" in terms of completeness and consistency. Thus, its use for supplementing search terms in information retrieval systems has led to disappointing results [13]. It may be more useful for providing suggestions or asking questions in an online context.

The EDR dictionary, now administered by the Japanese National Institute of Information and Communications Technology, provides an impressive range of lexical resources [14]. One of its main purposes is to support the development of robust and effective machine translation tools. Primary lexical access is via word dictionaries in Japanese and English, and these include links to a "concept dictionary" containing semantic information and hierarchical links to related concepts. Other components include co-occurrence dictionaries and a Japanese/English bilingual dictionary.

A thesaurus or online dictionary provides a kind of sketchy model of the world (or partial world), and the language used to describe it. The most fully developed systems of this kind are *ontologies* which, besides the features outlined above, may include definitions of processes, interactions, constraints, and temporal relationships [11].

Construction of Lexicons

Any body of text can be converted automatically into an *index* of words (or, with a little more trouble, of phrases) by extracting the words/phrases as discussed earlier and organizing them into an alphabetical list. An index thus produced is of course a kind of lexicon – one in which the outputs are restricted, e.g., to positional information or frequency. Many language processing applications start off by generating an index of extracted terms, which can then be inspected or manipulated at a later stage. Note that an inverted file is a form of index.

The compilation of more complex forms of lexicon can only be automated to a limited extent. Statistical analysis of text corpora can provide useful lexical resources, or at least resources which can be used to facilitate the construction of practically useful lexicons [9]. A key activity is looking for associations (collocations) between words in a text corpus [3]. This involves taking the corpus, dividing it into suitable segments (e.g., paragraphs, sentences, or fixed-size windows) and then measuring the extent to which different terms tend to occur together using, e.g., the mutual information measure. One application of this approach is in the construction of a *statistical thesaurus*, by which strongly associated terms may be used for expanding sets of query terms in information retrieval systems [1].

In a statistical thesaurus, the associations between terms are undifferentiated "tending-to-occur-together" relationships. More detailed analysis of patterns of word association may be used, tentatively, to differentiate between relationships of various types. One noteworthy example of this approach is the work of Greffenstette [7], while some further efforts in this direction have been reviewed by Matsumoto [11] and Srinivasan [6].

Key Applications

As has already been noted, some kind of lexical analysis is performed by almost every program which operates on textual data, including systems for text retrieval, question-answering, summarization, information extraction, data mining, and machine translation.

Data Sets

The URLs given below provide datasets as well as lexical analysis tools.

URL to Code

EDR Electronic Dictionary: http://www2.nict.go.jp/r/r312/EDR/index.html

Longman's Dictionary (LDOCE): http://www.ldoceonline.com/

Regular expressions: http://www.regular-expressions.info/

WordNet: http://wordnet.princeton.edu/

Cross-references

- ▶ Index Creation and File Structures
- ▶ Stemming
- ▶ Stoplists
- ▶ Query Expansion for Information Retrieval

Recommended Reading

1. Chen H., Schatz B., Yim T., and Fye D. Automatic thesaurus generation for an electronic community system. J. Amer. Society for Inform. Science, 46(3):175–193, 1995.
2. Chodorow M., Byrd R., and Heidorn, G. Extracting semantic hierarchies from a large on-line dictionary. In Proc. 23rd Annu. Meeting of the Association for Computation Linguistics, Illinois, USA, 1985, pp. 299–304.
3. Church K. and Hanks P. Word association norms: mutual information and lexicography. Computational Linguistics, 16(1): 22–29, 1990.
4. Clarke C.L.A. and Cormack G.V. On the use of regular expressions for searching text. ACM Transactions on Programming Languages and Systems, 19(3):413–426,1997.
5. Fellbaum C. (ed.). WordNet: An Electronic Lexical Database, MIT Press, Cambridge, MA, 1998.
6. Frakes W.B. and Baeza-Yates R. Information Retrieval: Data Structures & Algorithms, Chapters III, VII, and IX. Prentice-Hall, 1992.
7. Greffenstette G. Explorations in Automatic Thesaurus Discovery, Boston: Kluwer, 1994.
8. Greffenstette G. Tokenization, in H. van Halteren (ed.). Syntactic Wordclass Tagging, Kluwer, The Netherlands, 1999, pp. 117–133.
9. McEnery T. and Wilson A. Corpus Linguistics. Edinburgh University Press, UK, 2001.
10. Miller G.A., Beckwith R., Fellbaum C., Gross, D., and Miller K.J. Introduction to WordNet: an on-line lexical database. Int. J. Lexicography, 3:235–244, 1990.
11. The Oxford Handbook of Computational Linguistics. *1*Chapters III, XXI, XXIV, XXV, and XXXIII. Mitkov R. (ed.). Oxford University Press, UK, 2003.
12. Schneider J.W. and Borland P. Introduction to bibliometrics for construction and maintenance of thesauri: methodical considerations. J. Doc. 60(5):524–549, 2004.
13. Voorhees E.M. Using WordNet to disambiguate word senses for text retrieval. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 171–180.
14. Yokoi T. The EDR electronic dictionary. Commun. ACM, 38(11): 42–44, 1995.

Lexical Processing

- ▶ Lexical Analysis of Textual Data

Lexical Relations

- ▶ Term Proximity

Library of Congress METS

- ▶ LOC Mets

License

- ▶ Copyright Issues in Databases
- ▶ European Law in Databases
- ▶ Licensing and Contracting Issues in Databases

Licensing and Contracting Issues in Databases

Michael W. Carroll
Villanova University School of Law, Villanova, PA, USA

Synonyms

License

Definition

A contract is a legally binding agreement between two or more parties usually formed when each party promises to do some action not otherwise required or to refrain from taking some action not otherwise required. In this context, a license is a grant of rights or permission by the owner of rights under copyright in a database. The key differences between a copyright license and a contractual database license are who is subject to the license

and the consequences of breaching the license. A breach of a copyright license usually exposes the breaching party to liability for copyright infringement, which can include remedies such as injunctions, statutory damages, and disgorgement of profits. In contrast, a breach of contract exposes the breaching party to liability for the database owner's actual damages, which must be proven with reasonable certainty. Oftentimes, legal agreements styled as "terms of use" or as licenses in relation to databases are technically only partially copyright licenses and are otherwise merely contracts. This is because such agreements often contain terms governing the use of copyrightable elements of a database – such as its structure – and non-copyrightable data, such as numerical or other factual information. One who copies an entire database without permission would likely face liability for copyright infringement; whereas, one who copies factual data only without permission would be liable only for breach of contract.

Historical Background

Contract law has ancient roots, but it evolved into a general field relatively recently. In "common law" countries, such as the United States, the United Kingdom, Canada, and Australia, the law of contract is defined and applied by the courts. In "civil law" countries, which comprise the large majority of nations, the principles of contract are codified by the legislature and applied by the courts. Under both systems of law, a contract is enforceable if there has been an offer, an acceptance, and an exchange or "consideration," which means that each party has agreed to give the other something of value or has promised to for bear from taking action that would otherwise be lawful.

Foundations

To fill perceived gaps in copyright law, some database owners also have come to require agreement to contracts or license agreements in exchange for access to their databases. Three sets of legal issues arise with respect to these licenses: (i) what are the requirements to make them enforceable; (ii) are they enforceable if they control uses of non-copyrightable data; and (iii) what law applies if there are disputes about interpretation or enforcement of the license.

In the United States, a contract is enforceable if there is an offer, an acceptance, and consideration, which means that the parties have entered into a bargained-for-exchange. If the terms of a license are that the database owner agrees to make the database accessible and usable in exchange for the user's promises to pay for access or to refrain from making certain uses of the data – such as redistributing or copying data, the contract will be enforceable.

Outside of the United States, the requirement of exchange is supplemented by other methods of enforcement such that database licenses will generally be held enforceable under the applicable law of contract or obligations.

The key differences between a copyright license and a contractual database license are who is subject to the license and the consequences of breaching the license. Copyright is a right against the public, and a copyright owner may therefore grant the public a license to use the copyrighted elements of a database subject to a public license, such as a Creative Commons license. Most copyright licenses are drafted to say that if the user breaches, the copyright license terminates and any further use of the copyrighted elements becomes copyright infringement. The remedies for copyright infringement are more severe than for breach of contract.

A contractual database license is enforceable only against one who is a party to the agreement. If the license is from party *A* to party *B*, and party *C* obtains the database or data from party *B*, party *A* cannot enforce the contract against party *C*. To avoid this result, owners of digital databases often include terms in the database to the effect that any person who uses the database becomes a party to the license agreement.

There is some doubt about whether this approach comports with traditional requirements for enforceable contracts. Nonetheless, with respect to "terms of use" on web sites housing electronic databases courts have relaxed the requirement of exchange and have held that users accept such terms through the act of visiting or using the site.

With respect to whether a database owner may control uses of non-copyrightable data or databases by a contractual license when copyright law permits such uses, the question is one of preemption. Does contract law, which is a matter of state law in the United States, conflict with federal copyright law? If there is a conflict, federal law prevails. However, the few courts that have addressed this issue have reasoned that such licenses are not in conflict with copyright because the license applies only to the other party whereas copyright applies to the whole world.

Finally, the issue of which law governs the interpretation and enforcement of a contractual database license is an example of a choice-of-law issue. Often the license will specify what law applies, and courts will generally accept this term as long as the jurisdiction chosen has a nexus with the parties. Otherwise, the question of choice of law is determined through fact-specific balancing of the location of the parties, the place where the contract was formed, and, in the case of a digital library, the location of the server(s).

Key Applications

After the United States Supreme Court held in 1991 that a white pages telephone directory was not copyrightable, a case arose in which a database of more than 3,000 white pages directories had been compiled and released on CD-ROM with a license agreement that restricted those who purchased it to non-commercial uses. The defendant copied the data and resold it through his web site. The question presented was whether the non-commercial use restriction in the license agreement was enforceable even though the data was not protected by copyright. The court of appeals held that the agreement was enforceable. The court held that there was no conflict between the contract's restriction on copying the data and copyright law's position that the data could be freely copied because the contract only restricted those who agreed to its terms. The court reached this result only with respect to a particular section of the Copyright Act and did not address the argument that the constitution also preempted enforcement of the license agreement.

Other courts have largely followed this approach of enforcing similar agreements referred to as "shrinkwraps," when the license terms are inside a shrinkwrapped box of software; "clickwraps," where a user clicks an "I Agree" button to signal agreement to license terms; and, more controversially, to "browsewraps," in which the issue is whether a user of a web site has agreed to its terms of use merely by browsing the site. In this last category, the courts reach different results depending upon whether the user had adequate notice of the terms of use and whether those terms of use were reasonable.

Cross-references

- ▶ Copyright Issues in Databases
- ▶ European Law in Databases

Recommended Reading

1. Association of Research Libraries, Principles for Licensing Electronic Resources at http://www.arl.org/sc/licensing/licprinciples.shtml.
2. Bowers V. Baystate Techs., 320:F.3d 1317, 1320, Fed. Cir. 2003.
3. Greater Western Library Alliance Guidelines for Licensing Electronic Information Resources at http://www.gwla.org/reports/licensing.html.
4. Mark A. Lemley. Terms of use. Univ. of Minnesota Law Rev., 91:459–483, 2006.
5. ProCD V. Zeidenberg, 86:F.3d 1447, 7th Cir. 1996.

Lifespan

Christian S. Jensen
Aalborg University, Aalborg, Denmark

Synonyms

Existence time; Temporal domain

Definition

The *lifespan* of a database object is the time during which the corresponding real-world object exists in the modeled reality.

Key Points

Some temporal data models, e.g., conceptual models, provide built-in support for the capture of lifespans, while other models do not. It may be observed that lifespans can be reduced to valid times, in the sense that the lifespan of an object *o* is the valid time of the fact "*o* exists."

In the general case, the lifespan of an object is an arbitrary subset of the time domain and is naturally captured by a temporal element timestamp.

The synonym "existence time" is also used for this concept.

Cross-references

- ▶ Temporal Conceptual Models
- ▶ Temporal Database
- ▶ Temporal Element
- ▶ Time Domain
- ▶ Time in Philosophical Logic
- ▶ Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.), Böhlen M., Clifford J., Elmasri R., Gadia S. K., Grandi F., Hayes P., Jajodia S., Käfer W., Kline N., Lorentzos N., Mitsopoulos Y., Montanari A., Nonen D., Peressi E., Pernici B., Roddick J.F., Sarda N.L., Scalas M.R., Segev A., Snodgrass R.T., Soo M.D., Tansel A., Tiberio R., and Wiederhold G., A consensus glossary of temporal database concepts – February 1998 Version, in Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, and S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin Heidelberg New York, 1998, pp. 367–405.

Life-span (in Part)

► Context

Lightweight Ontologies

Fausto Giunchiglia, Ilya Zaihrayeu
University of Trento, Trento, Italy

Synonyms

Controlled vocabularies; Taxonomies; Thesauri; Business catalogues; Faceted classifications; Web directories; Topic hierarchies; User classifications

Definition

Ontologies are *explicit specifications of conceptualizations* [7]. They are often thought of as directed graphs whose nodes represent *concepts* and whose edges represent *relations* between concepts. The notion of a concept is understood as defined in Knowledge Representation, i.e., as a set of objects or individuals [2]. This set is called the *concept extension* or the *concept interpretation*. Concepts are often *lexically defined*, i.e., they have natural language names which are used to describe the concept extensions (e.g., concept `mother` denotes the set of all female parents). Therefore, when ontologies are visualized, their nodes are often shown with corresponding natural language concept names. The backbone structure of the ontology graph is a taxonomy in which the relations are "is-a," whereas the remaining structure of the graph supplies auxiliary information about the modeled domain and may include relations like "part-of," "located-in," "is-parent-of," and many others [8].

In their simplest version, one can think of *lightweight ontologies* as ontologies consisting of backbone taxonomies only. In this entry, the "is-a" relationship is generalized to concept subsumption which still allows it to respect the intrinsic properties of backbone taxonomies: namely, in a lightweight ontology, the extension of the concept of a child node is a subset of the extension of the concept of the parent node. Formally, the notion of lightweight ontology is defined as:

A *(formal) lightweight ontology* is a triple $O = \langle N, E, C \rangle$, where N is a finite set of nodes, E is a set of edges on N, such that $\langle N, E \rangle$ is a rooted tree, and C is a finite set of *concepts* expressed in a *formal language* F, such that for any node $n_i \in N$, there is one and only one concept $c_i \in C$, and, if n_i is the parent node for n_j, then $c_j \sqsubseteq c_i$.

The formal language F, used to encode concepts in C, belongs to the family of Description Logic (DL) languages [2] and it may differ in its expressive power and reasoning capability. However, the less expressive one with still useful reasoning capability has shown to be the *propositional DL language*, i.e., a DL language without roles (see [6,3,5] for examples of practical applications of formal lightweight ontologies based on the propositional DL language).

Taxonomies (e.g., NCBI (See http://www.ncbi.nlm.nih.gov/taxonomy)), thesauri (e.g., GLIN (See http://www.loc.gov/lexico/servlet/lexico.)), business catalogues (e.g., UNSPSC (See http://www.unspsc.org.)), faceted classifications (e.g., Flamenco (See http://flamenco.berkeley.edu.)), web directories (e.g., Yahoo! (See http://www.yahoo.com.)), and user classifications are examples of informal prototypes of formal lightweight ontologies. Hereinafter, they are referred to as *(informal) lightweight ontologies*. Note that lightweight ontologies are easier to understand and construct. In fact, as shown in [15], formal lightweight ontologies can be automatically constructed from user classifications as a by-product of a normal computer use, whereas designing a full-fledged ontology (expressed, for example, in OWL-DL (See http://www.w3.org/TR/owl-features.)) is a difficult and error-prone task even for experienced users [13].

Historical Background

The notion of ontology was borrowed from philosophy and adopted in Computer Science under multiple definitions, where, probably, the first credible and most commonly quoted one is "an explicit specification of a

conceptualization" [7]. As ontology research evolved, new definitions and examples of what can be considered to be an ontology started to appear. It is typical to characterize ontologies based on the degree of formality and expressivity of the language used to describe them. This characteristic form a continuum of ontology types (see Fig. 1), starting from terms and web directories, and continuing to rigorously formalized logical theories [14]. However, most specifications agree that an ontology should be defined in a formal language, which, in practice, usually means a logic-based language suitable for automating reasoning.

For a long time, lightweight ontologies were not formally defined but were referred to by examples. For instance, they were referred to as terms, as controlled vocabularies, as thesauri, and as web directories like Yahoo! [14]. As observed, informal lightweight ontologies largely cover the spectrum of informal ontology types shown in Fig. 1. In some other approaches, lightweight ontologies are formal ontologies which use a computationally inexpensive logic language (e.g., see [3,10,11]). In practice, these ontologies often encode a hierarchy of classes which can be (automatically or semi-automatically) derived from web directories like Yahoo! (as in [3,11]) or from more strictly defined but still informal structures such as thesauri and taxonomies (as in [10]).

The first attempt to formally define the notion of lightweight ontology as a type of formal ontology was made in [3]. Here, the definition was restricted to formal web directories, whereas in the present document it has been generalized to provide a formal model for a larger spectrum of informal lightweight ontologies, such as controlled vocabularies, taxonomies, thesauri, business catalogues, faceted classifications, web directories, and user classifications.

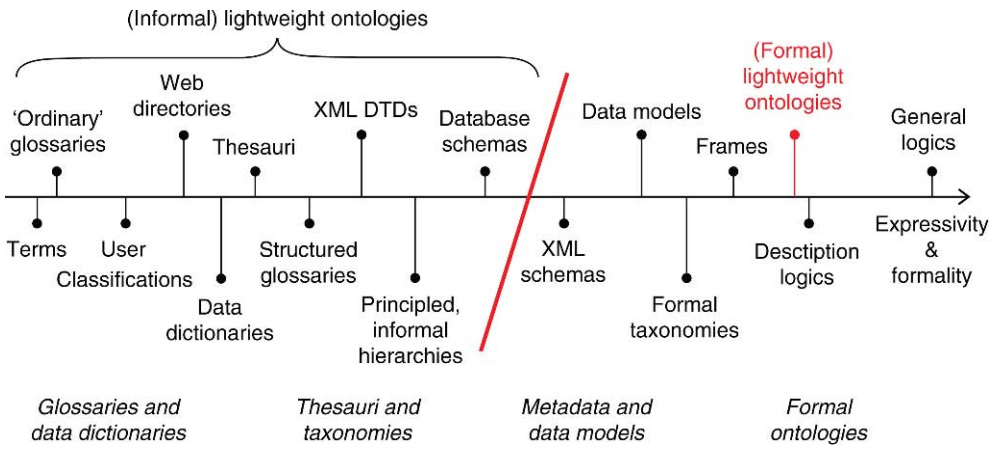
Foundations

This section discusses the main types of lightweight ontologies and their properties and proposes how formal lightweight ontologies can be generated from their informal prototypes.

Types of Lightweight Ontologies

Lightweight ontologies fall into two main types based on their usage: *descriptive* and *classification* lightweight ontologies. Descriptive lightweight ontologies are primarily used for defining the meaning of terms as well as the nature and structure of a domain [9]. Classification lightweight ontologies are primarily used for describing, classifying, and accessing (large) collections of documents or, more generally, data items [9,3]. Due to this difference, formal classification lightweight ontologies have a different domain of interpretation for their concepts. Namely, the extension of a concept in a formal classification lightweight ontology is the *set of documents about* the objects or individuals referred to by the (lexically defined) concept. For example, the extension of concept `mother` is the set of documents about female parents.

Any descriptive lightweight ontology can be used as a classification lightweight ontology, but not vice versa. The two types differ in some principal properties, which are summarized in Table 1 and discussed as follows. Classification lightweight ontologies are usually more complex than descriptive ones, whereas



Lightweight Ontologies. Figure 1. Types of ontologies. Adopted from [14].

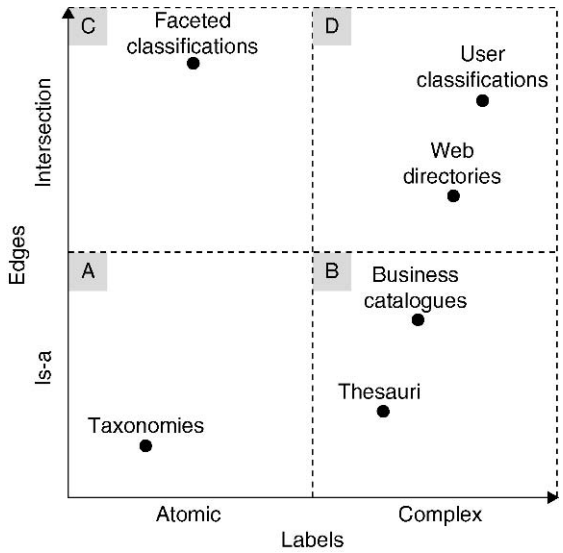
Lightweight Ontologies. Table 1. A classification of lightweight ontologies

	Descriptive lightweight ontologies	Classification lightweight ontologies
	Informal	
Primary use	Defining the meaning of terms as well as the nature and structure of a domain	Describing, classifying, and accessing (large) collections of documents or, more generally, data items
Labels	Single nouns or simple noun phrases denoting atomic concepts as the most typical case	Often, compound noun phrases denoting complex concepts
Edge relations	"is-a" relation	"is-a," "part-of," or, more generally, "intersection" relation
Examples	Taxonomies (e.g., NCBI), thesauri (e.g., GLIN)	Business catalogues (e.g., UNSPSC), faceted classifications (e.g., Flamenco), web directories (e.g., Yahoo!), user classifications
	Formal	
Concept extension	Set of objects or individuals belonging to the class denoted by the concept	Set of documents about the objects or individuals belonging to the class denoted by the concept
Node concepts	Atomic concepts	Atomic and complex concepts

the complexity is defined along two dimensions: *label complexity* (atomic vs. complex labels) and *edge complexity* ("is-a" vs. "intersection" edges). The dimensions of complexity are illustrated in Fig. 2 and discussed in the following.

Category A: atomic labels and "is-a" edges. Ontology labels in this category usually represent single atomic concepts (e.g., "theft," "cellular organisms") and relations between labels are usually "is-a" relations (e.g., "pens" is a child of "writing materials"). Typical examples of this category are (biological) taxonomies such as NCBI. Ontologies in this category are mainly descriptive.

Category B: complex labels and "is-a" edges. Ontology labels in this category can be compound noun phrases which represent complex concepts (e.g., "Open Source and Linux in Education," "Pressure groups representation or participation services") and relations between labels are usually "is-a" relations. Typical examples of this category are thesauri such as GLIN and business catalogues such as UNSPSC. A higher complexity of labels (with regard to category A) in these domains is conditioned by the need for richer descriptions of indexing terms in thesauri and of e-commerce items in business catalogues. Most ontologies in this category are descriptive but some can be classification as well. For instance, the business catalogue UNSPSC can be used as a descriptive ontology or as a classification ontology in which e-commerce items are classified. Note that even if the labels can be



Lightweight Ontologies. Figure 2. Types of labels and edges in lightweight ontologies.

complex, they are still mapped to atomic concepts in formal descriptive lightweight ontologies. In classification lightweight ontologies, complex labels represent a dimension of *power of classification* as one label can describe one complex concept that identifies a (very) specific set of documents. Moreover, complex labels can be mapped into complex concepts in formal classification ontologies, which allows for higher modularity in concept definitions. For instance, concept `baby_pictures` can be defined as the intersection

of two concepts, `baby` and `picture`, whereas the interpretation of the former concept is the set of documents about babies (including pictures of babies as a kind of documents) and the interpretation of the latter concept is the set of pictures (including baby pictures). Note that in formal descriptive lightweight ontologies, the extension of concept `baby_pictures` (the set of baby pictures in the world) cannot be expressed as a function of the extension of concept `baby` (the set of babies in the world) and the extension of concept `picture` (the set of pictures in the world).

Category C: atomic labels and "intersection" edges. Ontology labels in this category usually represent single atomic concepts, and relations between labels are usually "intersection" relations, which means that the label of a parent node *specifies* the meaning of the label of its child node. For example, parent node "Italy" specifies the meaning of its child node "Vacation" to the meaning "Vacation in Italy." A typical example of this category is a faceted classification such as Flamenco, in which child nodes represent aspects or facets of their parent nodes along atomic orthogonal dimensions (e.g., time, space, function, material, etc). All ontologies in this category are classification ontologies, for which the "intersection" relation creates an additional dimension of power of classification by allowing it to describe a specific set of documents by means of a set of categories in the ontology. Note that the interpretation domain of formal classification ontologies allows it to treat edges as the *intersection* of parent and child concepts and, therefore, compute concepts of nodes given their position in the ontology tree. For example, the intersection of root concept `italy` with its child concept `vacation` results in a concept whose extension is the set of documents about vacations in Italy, which is the actual meaning of the child node, given its position in the tree.

Category D: complex labels and "intersection" edges. Ontology labels in this category usually represent complex concepts, and relations between labels are usually "intersection" relations. All ontologies in this category are classification ontologies for which the combination of complex labels and "intersection" edges creates the maximum classification power. Labels in this category can represent names of individuals. These labels are mapped to concepts whose extension is the set of documents about the individuals (e.g., the extension of concept `moscow` is the set of documents about the city Moscow). Typical examples of this category are web directories like Yahoo! (in which web pages are classified) and user classifications (in which email messages, favorites, and files are classified). Note that user classifications may have more complex labels and more "intersection" relations than web directories due to the fact that there are basically no rules and restrictions for user classifications that are commonly followed in web directories.

Note that the propositional DL is sufficient for representing formal descriptive lightweight ontologies in categories A and B, as the only thing that needs to be encoded is the subsumption hierarchy of atomic classes. It is also sufficient for representing formal classification lightweight ontologies in categories A and C, since the only relations that need to be represented are subsumption and intersection of atomic classes. The propositional DL is capable of capturing the semantics of labels in classification lightweight ontologies in categories B and D to a significant extent, while approximating the meaning of labels in some cases, as it is discussed in the following section.

From Informal to Formal Lightweight Ontologies

Formal descriptive lightweight ontologies can be generated from informal ones by transforming their organizational structure into a rooted tree (where necessary) and by converting their term labels, expressed in natural language, into concepts in the formal language *F*. The generation of formal classification lightweight ontologies requires an extra step where node concepts are computed as the intersection of the concepts corresponding to the term labels on the path to the root node [3]. Note that the rooted tree structure of formal lightweight ontologies allows it to capture the backbone organization of many informal prototypes. In fact, taxonomies, business catalogues, faceted classifications, web directories, and user classifications use rooted trees to organize their categories. In the simplest case, the hierarchy of thesaurus terms, built based on the Broader Term relation, is a rooted tree; and, a controlled vocabulary can be seen as one level rooted tree, where the root node represents the Top concept and its child nodes are the controlled vocabulary terms. The principal steps of the process of converting term labels into concepts in *F* for classification lightweight ontologies are described below. The conversion of labels of descriptive lightweight ontologies follows the same principles even if it is an easier case due to the relative simplicity of their labels. In the following, it is assumed that *F* is the Propositional DL language.

WordNet [12] senses of adjectives and common nouns in the label become atomic concepts in *F*. The extension of a common noun concept is the set of documents about objects of the class, denoted by the noun; and, the extension of an adjective concept is the set of documents about objects, which possess the qualities, denoted by the adjective. Proper names become atomic concepts as well, whose extension is the set of documents about the individual referenced by the proper name. Notationally, adjective and common noun atomic concepts are represented in the following syntax: *lemma-pos-sn*, where *lemma* is the lemma of the word, *pos* is its part of speech, and *sn* is the sense number in WordNet [12]. Proper name atomic concepts are tagged with $_{NNP}$.

Syntactic relations between words in the label are translated into logical connectives of *F* in order to build complex concepts from atomic concepts. For example, a set of adjectives followed by a noun group is translated into the logical conjunction ($\sqcap$) of the concepts corresponding to the adjectives and to the nouns; prepositions like "of" and "in" are translated into the conjunction; coordinating conjunctions "and" and "or" are translated into the logical disjunction ($\sqcup$). The final formula for the label is built following these rules and taking into account how words are coordinated in the label.

Consider a relatively complex label: "*Bank and personal details of George Bush*." Its correct translation to *F* produces the following concept:

► $(\texttt{bank-noun-1} \sqcup \texttt{personal-adj-1}) \sqcap \texttt{detail-noun-1} \sqcap \texttt{george\_bush}_{\texttt{NNP}}$

The extension of the concept above is the intersection of three sets of documents: (1) documents about the President George W. Bush, (2) documents containing isolated facts about something (i.e., details), and (3) the union of documents about bank institutions and documents concerning a particular person or his/her private life. Note that the extension comprises (all and only) documents one would classify under the above mentioned label.

Despite its seeming simplicity, the translation process is subject to various mistakes originating from incorrect natural language processing (NLP). (How the NLP problems, described in this paragraph, can be solved is beyond the scope of this document. Interested readers are referred to [15] for a first account.) For instance, due to a mistake in part-of-speech tagging, the word *personal* might be recognized as a noun, which has only one sense in WordNet defined as "*a short newspaper article about a particular person or group*." Due to a mistake in word sense disambiguation, the sense of the word *bank* might be identified as "*sloping land (especially the slope beside a body of water)*." Due to a mistake in named entity detection, the proper name *George Bush* might not be detected and might then be considered as two distinct nouns, where the noun *bush* means "*a low woody perennial plant usually having several major branches*." Finally, due to a mistake in (syntax) parsing, the input label might be erroneously translated into:

► $\texttt{bank-noun-1} \sqcup \texttt{personal-adj-1} \sqcap \texttt{detail-noun-1} \sqcap \texttt{george\_bush}_{\texttt{NP}}$

a concept, whose extension is the union of documents about bank institutions and documents discussing personal details of the President George W. Bush.

Note that the propositional DL can capture the semantics of complex labels to a significant extent only when these labels are built from noun phrases possibly connected through coordinating conjunctions such as "and" and "or" (e.g., "Big city life and civil protection"). It approximates the meaning of some other types of labels, e.g., labels with prepositions. For instance, labels "life in war" and "life after war" will collapse into the same formula, which approximates the meaning of both labels.

Key Applications

Formal (descriptive and classification) lightweight ontologies can be used in various domains, such as document classification (e.g., see [6]), semantic search (e.g., see [3]), and data integration (e.g., see [5]). In the following subsections these three application domains are briefly discussed.

Document Classification

Document classification is the problem of assigning a document to one or more categories based on the document contents. In the context of lightweight ontologies, document classification refers to assigning a document to: (1) controlled vocabulary terms; (2) categories in taxonomies, business catalogues, faceted classifications, web directories, or user classifications. The approach reported in [6] presents fully automatic classification of documents into web directories based on the *get-specific* document classification algorithm.

The underlying idea is that a web directory is converted into a formal lightweight ontology, that a document is assigned a concept, and that the document classification problem is then reduced to reasoning about subsumption on the formal lightweight ontology. Note that this classification approach does not require the creation of a training dataset which would normally be required in machine learning approaches [6].

Semantic Search

In the context of lightweight ontologies, semantic search is the problem of finding categories and/or documents (when applicable) classified in categories of (informal) lightweight ontologies, such that the found objects *semantically* correspond to a provided natural language query. Loosely speaking, semantic correspondence of an object to a query indicates that the meaning associated with the object is more specific or equivalent to the meaning given to the query under common sense interpretation. For instance, document about *Ethiopian villages* semantically corresponds to a query about *African settlements*. The approach reported in [3] formalizes the above informal description and introduces a semantic search algorithm for lightweight classification ontologies populated with documents. The underlying idea is that the user query is converted to a concept in the way presented earlier in this document and that the answer to the query is computed as the set of documents whose concepts are more specific or equivalent to the concept of the query. In order to reduce the computation complexity, the query is first run on the structure of the corresponding formal lightweight ontology in order to identify the scope of relevant nodes and then it is run on the documents populated in some of the nodes from the scope.

Data Integration

Data integration is the process of combining data residing at different sources and providing the user with a unified view of these data. Often, a data source can be represented in the form of a rooted tree, whose nodes are assigned natural language labels, and, in this case, data integration can be facilitated by discovering semantic relations which exist between nodes of the source trees [5]. A semantic relation between two nodes can be more/less general, equivalent, or disjoint. In the domain of lightweight ontologies, semantic relations can be found between elements of controlled vocabularies, taxonomies, thesauri, business catalogues, faceted classifications, web directories, and user classifications. Such relations can then be used for enabling integration or inter-operation of web directories, for merging business catalogues, and so on.

Future Directions

There are two major problems related to formal lightweight ontologies which drive future directions of research in this area. The problems are:

- *Natural language processing.* Since formal lightweight ontologies are supposed to be often generated from their informal prototypes, the quality of these ontologies strongly depends of the correctness and completeness of NLP procedures involved in the conversion process. Note that NLP for informal lightweight ontologies is a potentially new domain in the NLP research due to the particular characteristics of term labels (e.g., they are usually short noun phrases with little context) [15];
- *Lack of background knowledge.* Reasoning on formal lightweight ontologies, which is used, for example, in document classification, in semantic search, and in data integration as discussed above, strongly depends on the set of axioms which must be known a priori [3]. These axioms are extracted from a knowledge base such as WordNet [12]. It was shown that lack of background knowledge is the main source of a relatively low recall in reasoning-based tasks on formal lightweight ontologies [4].

Experimental Results

First evaluation studies of document classification show that re-classification of 1217 HTML pages into a part of the DMoz hierarchy, (See http://www.dmoz.org.) that has 157 nodes distributed in a tree of depth 6, allows it to reach 41% in the micro-averaged F1 measure [6]. Document concepts in the conducted experiments were built by computing the conjunction of the formulas corresponding to the first ten most frequent words appearing in the documents (excluding stop words).

The performance of S-Match, a tool that, among other things, facilitates the integration of lightweight ontologies, reaches up to 65% in recall in some data sets [1].

Data Sets

Some informal lightweight ontologies are available for download in the form of data files. These include the

DMoz web directory, (See http://rdf.dmoz.org.) business catalogues UNSPSC and eCl@ss, (See http://www.eclassdownload.com.) NCBI taxonomy, and many others.

Cross-references

► I(a) – Database Fundamentals: Data Models (including semantic data models)
► Data Integration

Recommended Reading

1. Avesani P., Giunchiglia F., and Yatskevich M. A Large Scale Taxonomy Mapping Evaluation. In Proc. 4th Int. Semantic Web Conference, 2005, pp. 67–81.
2. Baader F., Calvanese D., McGuinness D., Nardi D., and Patel-Schneider P. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003.
3. Giunchiglia F.M.M. and Zaihrayeu I. Encoding classifications into lightweight ontologies. J. Data Semant., VIII:57–81, 2007.
4. Giunchiglia F., Shvaiko P., and Yatskevich M. Discovering Missing Background Knowledge in Ontology Matching. In Proc. 17th European Conf. on Artificial Intelligence, 2006, pp. 382–386.
5. Giunchiglia F., Yatskevich M., and Shvaiko P. Semantic Matching: Algorithms and Implementation. J. Data Semant., IX, 2007.
6. Giunchiglia F., Zaihrayeu I., and Kharkevich U. Formalizing the Get-Specific Document Classification Algorithm. In 11th European Conf. on Research and Advanced Technology for Digital Libraries, 2007.
7. Gruber T.R. A translation approach to portable ontology specifications. Knowl. Acquis., 5(2):199–220, 1993.
8. Guarino N. Some Ontological Principles for Designing Upper Level Lexical Resources. In Proc. 1st Int. Conf. Lexical Resources and Evaluation. vol. 2830, 1998.
9. Guarino N. Helping People (and Machines) Understanding Each Other: The Role of Formal Ontology. In Proc. CoopIS/DOA/ODBASE, 2004, p. 599.
10. Hepp M. and de Bruijn J. GenTax: A Generic Methodology for Deriving OWL and RDF-S Ontologies from Hierarchical Classifications, Thesauri, and Inconsistent Taxonomies. In Proc. 4th European Semantic Web Conference, 2007, pp. 129–144.
11. Magnini B., Serafini L., and Speranza M. Making Explicit the Hidden Semantics of Hierarchical Classifications. In Proc. 8th Congress of the Italian Association for Artificial Intelligence, 2003, pp. 436–448.
12. Miller G. Wordnet: An electronic Lexical Database. MIT Press, Cambridege, 1998.
13. Rector A.L., Drummond N., Horridge M., Rogers J., Knublauch H., Stevens R., Wang H., and Wroe C. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In Proc. 4th Int. Conf. Knowledge Eng. and Knowledge Management: Ontologies and the Semantic Web, 2004, pp. 63–81.
14. Uschold M. and Gruninger M. Ontologies and semantics for seamless connectivity. ACM SIGMOD Rec., 33(4):58–64, 2004.
15. Zaihrayeu I., Sun L., Giunchiglia F., Pan W., Ju Q., Chi M., and Huang X. From Web Directories to Ontologies: Natural Language Processing Challenges. In Proc. 6th International Semantic Web Conference, 2007.

Lineage

► Provenance
► Provenance in Scientific Databases

Linear Hashing

Donghui Zhang[1], Yannis Manolopoulos[2], Yannis Theodoridis[3], Vassilis J. Tsotras[4]
[1]Northeastern University, Boston, MA, USA
[2]Aristotle University, Thessaloniki, Greece
[3]University of Piraeus, Piraeus, Greece
[4]University of California – Riverside, Riverside, CA, USA

Definition

Linear Hashing is a dynamically updateable disk-based index structure which implements a hashing scheme and which grows or shrinks one bucket at a time. The index is used to support exact match queries, i.e., find the record with a given key. Compared with the B+-tree index which also supports exact match queries (in logarithmic number of I/Os), Linear Hashing has better expected query cost O(1) I/O. Compared with Extendible Hashing, Linear Hashing does not use a bucket directory, and when an overflow occurs, it is not always the overflown bucket that is split. The name Linear Hashing is used because the number of buckets grows or shrinks in a linear fashion. Overflows are handled by creating a chain of pages under the overflown bucket. The hashing function changes dynamically and at any given instant there can be at most two hashing functions used by the scheme.

Historical Background

A hash table is an in-memory data structure that associates keys with values. The primary operation it supports efficiently is a lookup: given a key, find the

corresponding value. It works by transforming the key using a hash function into a hash, a number that is used as an index in an array to locate the desired location where the values should be. Multiple keys may be hashed to the same bucket, and all keys in a bucket should be searched upon a query. Hash tables are often used to implement associative arrays, sets and caches. Like arrays, hash tables have O(1) lookup cost on average.

Foundations

The Linear Hashing scheme was introduced by [2].

Initial Layout

The Linear Hashing scheme has m initial buckets labeled 0 through $m - 1$, and an initial hashing function $h_0(k) = f(k) \% m$ that is used to map any key k into one of the m buckets (for simplicity assume $h_0(k) = k \% m$), and a pointer p which points to the bucket to be split next whenever an overflow page is generated (initially $p = 0$). An example is shown in Fig. 1.

Bucket Split

When the first overflow occurs (it can occur in any bucket), bucket 0, which is pointed by p, is split (rehashed) into two buckets: the original bucket 0 and a new bucket m. A new empty page is also added in the overflown bucket to accommodate the overflow. The search values originally mapped into bucket 0 (using function h_0) are now distributed between buckets 0 and m using a new hashing function h_1.

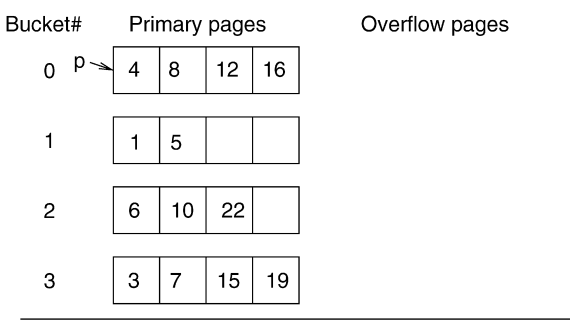
As an example, Fig. 2 shows the layout of the Linear Hashing of Fig. 1 after inserting a new record with key 11. The circled records are the existing records that are moved to the new bucket. In more detail, the record is inserted into bucket 11%4 = 3. The bucket overflows and an overflow page is introduced to accommodate the new record. Bucket 0 is split and the records originally in bucket 0 are distributed between bucket 0 and bucket 4, using a new hash function $h_1(k) = k \% 8$.

The next bucket overflow, such as triggered by inserting two records in bucket 2 or four records in bucket 3 in Fig. 2, will cause a new split that will attach a new bucket $m + 1$ and the contents of bucket 1 will be distributed using h_1 between buckets 1 and $m + 1$. A crucial property of h_1 is that search values that were originally mapped by h_0 to some bucket j must be remapped either to bucket j or bucket $j + m$. This is a necessary property for Linear Hashing to work. An example of such hashing function is: $h_1(k) = k \% 2m$. Further bucket overflows will cause additional bucket splits in a linear bucket-number order (increasing p by one for every split).

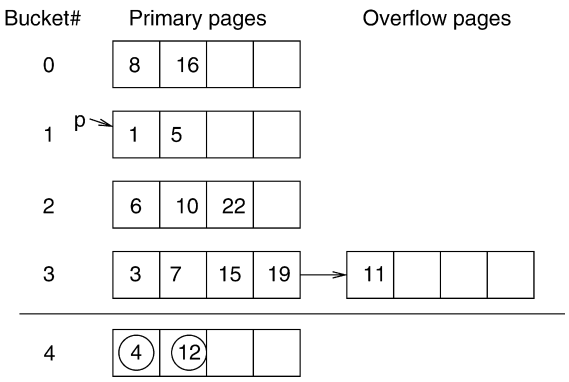
Round and Hash Function Advancement

After enough overflows, all original m buckets will be split. This marks the end of splitting-round 0. During round 0, p went subsequently from bucket 0 to bucket $m - 1$. At the end of round 0 the Linear Hashing scheme has a total of $2m$ buckets. Hashing function h_0 is no longer needed as all $2m$ buckets can be addressed by hashing function h_1. Variable p is reset to 0 and a new round, namely splitting-round 1, starts. A new hash function h_2 will start to be used.

In general, the Linear Hashing scheme involves a family of hash functions h_0, h_1, h_2, and so on. Let the



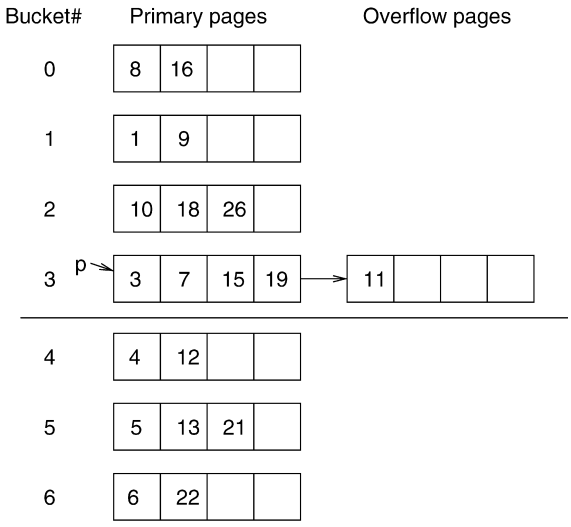
Linear Hashing. Figure 1. An initial Linear Hashing. Here $m = 4$, $p = 0$, $h_0(k) = k \% 4$.



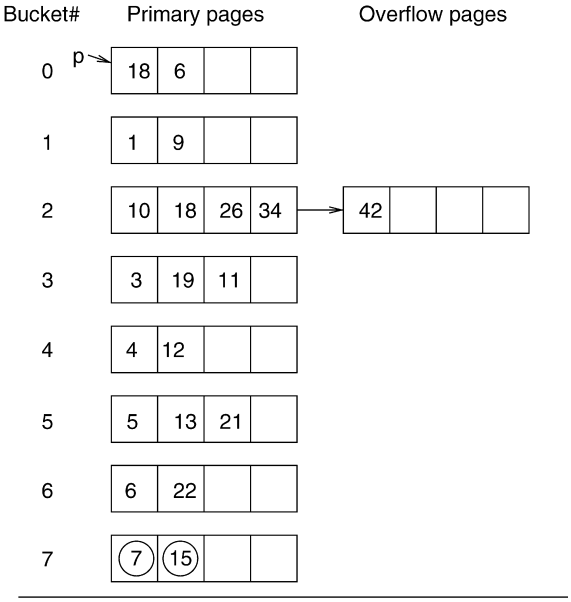
Linear Hashing. Figure 2. The Linear Hashing after inserting 11 into Fig. 1. Here $p = 1$, $h_0(k) = k \% 4$, $h_1(k) = k \% 8$.

initial function be $h_0(k) = f(k) \% m$, then any later hash function $h_i(k) = f(k) \% 2^i m$. This way, it is guaranteed that if h_i hashes a key to bucket $j \in [0..2^i m - 1]$, h_{i+1} will hash the same key to either bucket j or bucket $j + 2^i m$. At any time, two hash functions h_i and h_{i+1} are used.

Figures 3 and 4 illustrates the cases at the end of splitting-round 0 and at the beginning of splitting-round 1. In general, in splitting round i, the hash functions h_i and h_{i+1} are used. At the beginning of round i, $p = 0$ and there are $2^i m$ buckets. When all of these buckets are split, splitting round $i + 1$ starts. p goes back to 0. The number of buckets becomes $2^{i+1} m$. And hash functions h_{i+1} and h_{i+2} will start to be used.



Linear Hashing. Figure 3. The Linear Hashing at the end of round 0. Here $p = 3$, $h_0(k) = k \% m$, $h_1(k) = k \% 2^1 m$.



Linear Hashing. Figure 4. The Linear Hashing at the beginning of round 1. Here $p = 0$, $h_1(k) = k \% 2^1 m$, $h_2(k) = k \% 2^2 m$.

Component Summary and Search Scheme

In summary, at any time a Linear Hashing scheme has the following components:

- A value i which indicates the current splitting round.
- A variable $p \in [0..2^i m - 1]$ which indicates the bucket to be split next.
- A total of $2^i m + p$ buckets, each of which consists of a primary page and possibly some overflow pages.
- Two hash functions h_i and h_{i+1}.

A search scheme is needed to map a key k to a bucket, either when searching for an existing record or when inserting a new record. The search scheme works as follows:

(a) If $h_i(k) \geq p$, choose bucket $h_i(k)$ since the bucket has not been split yet in the current round.
(b) If $h_i(k) < p$, choose bucket $h_{i+1}(k)$, which can be either $h_i(k)$ or its spit image $h_i(k) + 2^i m$.

For example, in Fig. 2, $p = 1$. To search for record 5, since $h_0(5) = 1 \geq p$, one directly goes to bucket to find the record. But to search for record 4, since $h_0(4) = 0 < p$, one needs to use h_1 to decide the actual bucket. In this case, the record should be searched in bucket $h_1(4) = 4$.

Variations

A split performed whenever a bucket overflow occurs is an uncontrolled split. Let l denote the Linear Hashing scheme's load factor, i.e., $l = S/b$ where S is the total number of records and b is the number of buckets used. The load factor achieved by uncontrolled splits is usually between 50–70%, depending on the page size and the search value distribution [2]. In practice, higher storage utilization is achieved if a split is triggered not by an overflow, but when the load factor l becomes greater than some upper threshold. This is called a controlled split and can typically achieve 95% utilization. Other controlled schemes exist where a split is delayed

until both the threshold condition holds and an overflow occurs.

Deletions will cause the hashing scheme to shrink. Buckets that have been split can be recombined if the load factor falls below some lower threshold. Then two buckets are merged together; this operation is the reverse of splitting and occurs in reverse linear order. Practical values for the lower and upper thresholds are 0.7 and 0.9 respectively.

Linear Hashing has been further investigated in an effort to design more efficient variations. In [3] a performance comparison study of four Linear Hashing variations is reported.

Key Applications

Linear Hashing has been implemented into commercial database systems. It is used in applications where exact match query is the most important query such as hash join [4]. It has been adopted in the Icon language [1].

Cross-references

- ▶ Extendible Hashing
- ▶ Hashing
- ▶ Hash-based Indexing

Recommended Reading

1. Griswold W.G. and Townsend G.M. The design and implementation of Dynamic Hashing for sets and tables in icon. Software Pract. Ex., 23(4):351–367, 1993.
2. Litwin W. Linear Hashing: a new tool for file and table addressing. In Proc. of the Sixth International Conference on Very Large Databases, 1980, pp. 212–223.
3. Manolopoulos Y. and Lorentzos N. Performance of Linear Hashing schemes for primary key retrieval. Inf. Syst., 19(5):433–446, 1994.
4. Schneider D.A. and DeWitt D.J. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In Proc. 16th Int. Conf. on Very Large Databases, 1990, pp. 469–480.

Linear Regression

Jialie Shen
Singapore Management University, Singapore, Singapore

Definition

Linear regression is a classical statistical tool that models relationship between a dependent variable or regress and Y, explanatory variable or regressor $X = \{x_1,..,x_I\}$ and a random term ε by fitting a linear function,

$$Y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_I x_I + \varepsilon$$

where α_0 is the constant term, the α_is are the respective parameters of independent variable, and I is the number of parameters to be estimated in the linear regression.

Key Points

Linear regression analysis is an important component for several tasks such as clustering, time series analysis, and information retrieval. For instance, it is a very powerful forecasting method for time series data. It helps identify the long-term movement of a certain data set based on given information and explore the dependent variable as function of time.

To solve the linear regression problem, there are various kinds of approaches available to determine suitable regression coefficients [1,2]. They include: (i) least-squares analysis, (ii) assessing the least-squares model, (iii) modifications of least-squares analysis, and (iiii) polynomial fitting. The primary objective is to select a straight line which minimizes the error between the real data and the line estimated to provide a best fit.

Cross-references

- ▶ Least Squares
- ▶ Numeric Prediction
- ▶ Regression

Recommended Reading

1. Draper, N.R., Smith, H. Applied Regression Analysis Wiley Series in Probability and Statistics, 1998.
2. Gross, J. Linear Regression, Springer, Berlin, 2003.

Linearization

- ▶ Space Filling Curves
- ▶ Space-Filling Curves for Query Processing

Link Analysis

- ▶ Web Page Quality Metrics

Link Database

- ▶ Graph Database

Linked Brushing

► Linking and Brushing

Linked Views

► Linking and Brushing

Linking and Brushing

Matthew O. Ward
Worcester Polytechnic Institute, Worcester, MA, USA

Synonyms

Linked brushing; Linked views

Definition

Within the context of visual data exploration, *Linking* refers to the process in which user interactions in one display of a multi-display system are applied to some or all other displays. In this same context, *brushing* consists of the interactive selection of a subset of the displayed data by either dragging the mouse over the data of interest or using a bounding shape to isolate this subset. Together, linked brushing is one of the most powerful interactive tools for doing exploratory data analysis using visualization.

Historical Background

Perhaps the earliest reference to linked brushing was by McDonald [10] as a mechanism for cross-referencing between multiple plots. The term *brushing* was introduced in 1978 by Newton [11], who defined it as an interactive method for painting a group of points with a square, circular, or polygonal brush. Since then, researchers have expanded on these concepts, as described in the next section.

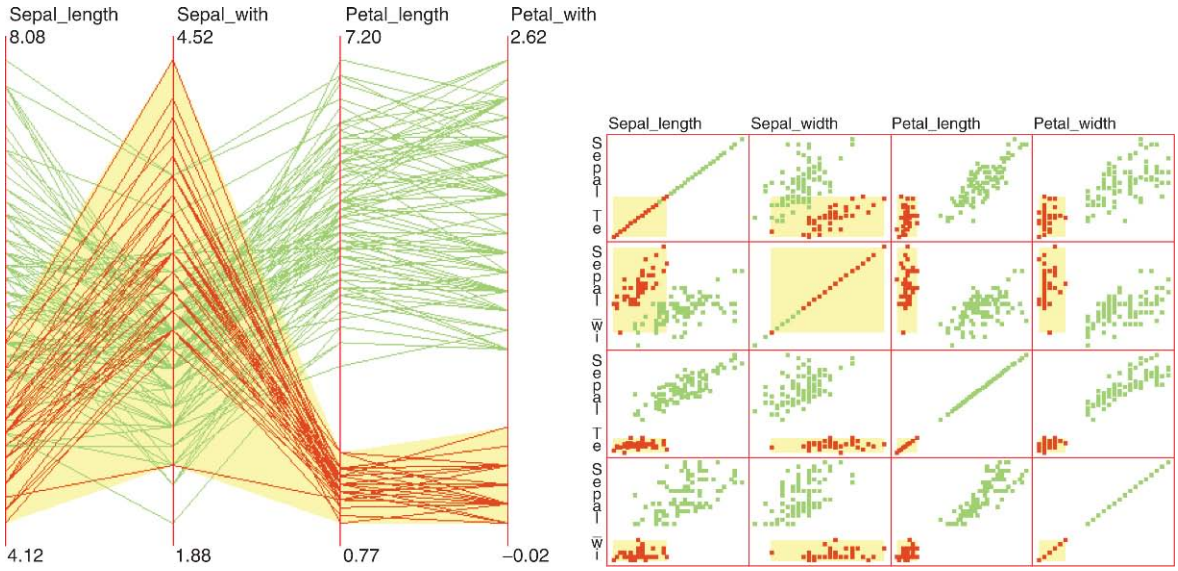
Foundations

Many operations within the process of visual data analysis begin with the process of *Selection*, in which a subset of the data being analyzed is isolated. Once isolated, the subset may be highlighted, deleted, masked, aggregated, or otherwise subjected to some operation. Many methods for performing interactive selection have been developed, including clicking on items in a list, specifying a range query for one or more data variables, typing a set of constraints, and sampling. These methods are sometimes referred to as *indirect*, as their specification is most often done separate from the data visualization. Indeed, most, if not all, can be performed without even viewing the data.

Brushing is a selection method that is specified in a *direct* fashion, i.e., by indicating elements or constraints on the data visualization itself. Many variations on brushing have been proposed over the years, each with its own strengths and weaknesses in terms of ease of use and degree of control a user has on the constitution of the result set. Brushing techniques usually consist of combinations of mouse/cursor motions and button clicks, though more unusual methods, such as using eye/head tracking or gestures in virtual reality environments, have also been proposed.

One key aspect that differentiates many brushing techniques is the *space* in which the interaction is specified. Early brushing implementations, e.g., [1], used screen-space as a constraint. Thus data that mapped to a particular range of pixels in the display were considered selected. Later, brushing was expanded to data-space [9], in which a selection from a multivariate data visualization consisted of a bounding box whose dimensionality matched the number of variables in the data (see Fig. 1). These N-dimensional brushes could be generated either by painting over a subsection of the display or directly manipulating a visual depiction of the boundaries of the N-dimensional hyperbox enveloping the selection. A third space that has been proposed for brushing is structure-space [6], where the structure of a data set, e.g., an ordering, hierarchy, or other organization, is used as a mechanism for data selection. In this case, a visual representation of the structure is provided to the user, and he or she brushes over parts of the structure. Data falling into this part are then considered as selected (see Fig. 2). This type of selection could conceivably be considered both direct and indirect, as the operation is performed on a visualization, but not on the data display itself. Similar to structure-based brushing, other attributes of a data set, such as the uncertainty or quality of the data [13], could be used as a focus for brushing.

Brushing techniques also differ in the manner in which data is isolated. Common methods include bounding boxes, bounding circles, lassos, and arbitrary



Linking and Brushing. Figure 1. Example of linked brushing in data space. Cluster isolated in parallel coordinates, with linked selection in scatterplot matrix. Selected data is dark, brush extents are shown as light bands or rectangles.

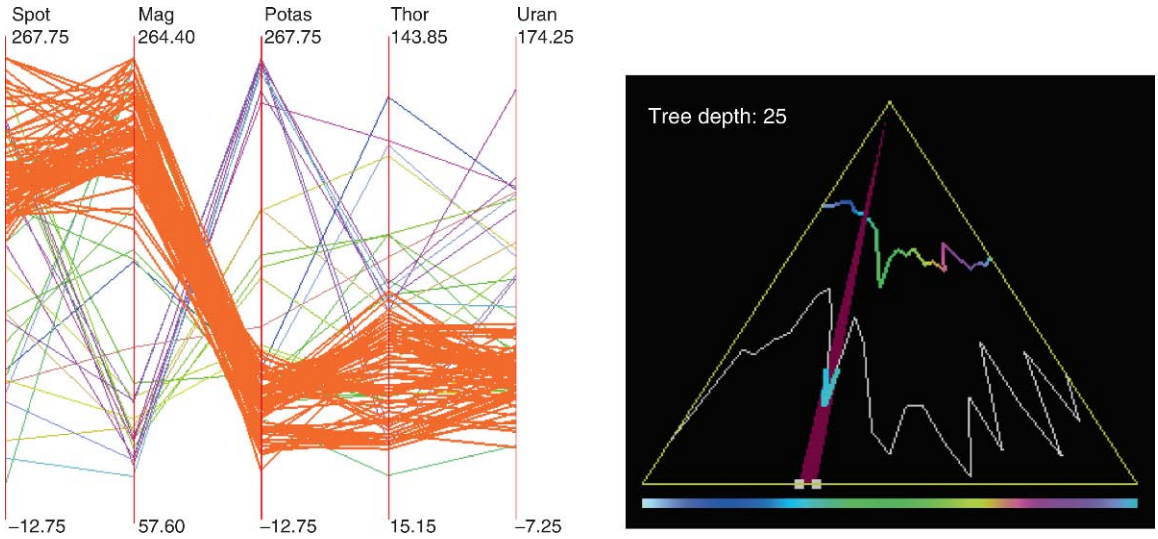
polygons for specifying a boundary around the selection and painting over specific data points. One can even specialize the behavior of a brush based on a specific type of visualization. Hauser et al. [7], for example, use an angled brush on parallel coordinates displays to select only data points whose representation on the display form lines with angles similar to that specified in the brush. In this way, data points that are well correlated with each other between a given pair of dimensions can be readily isolated.

Many approaches have been proposed to increase the richness of the types of selections that are possible. Martin and Ward [9] allow the user to not only control up to four separate brushes, but also allowed selections to be formed by logical combinations of brushes, including unions, intersections, negations, and exclusive or's. Wills [12] expanded on this idea by developing what might be called *incremental* brushes, where by a composite brush could be formed by augmenting an existing selection with additional data and/or constraints. This can result, for example, in selections with nearly arbitrary shapes, including holes.

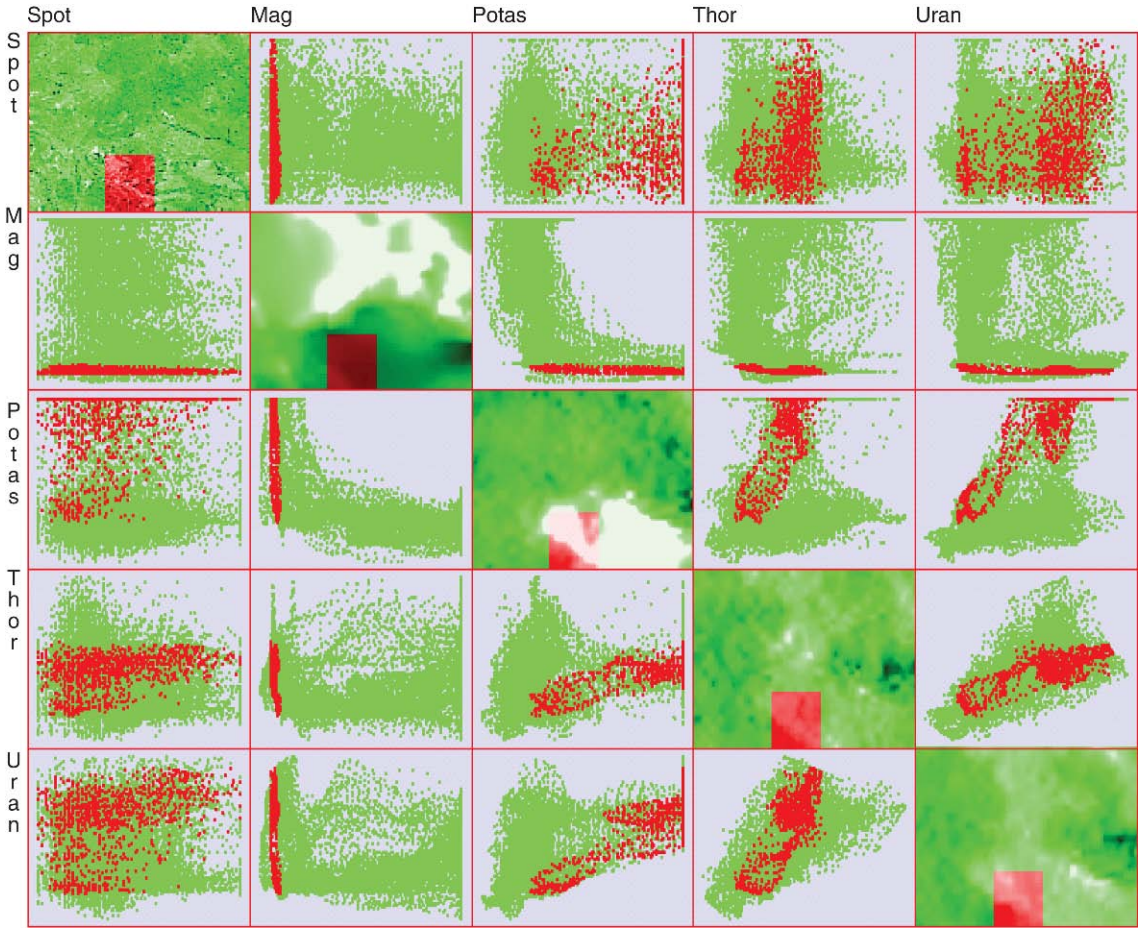
Thus far, the assumption has been that the result of brushing is that every data point in the set is either in the brush or outside the brush. However, some researchers [5,9] have experimented with what might be called *fuzzy* brushes, in which data can have a degree of membership in a brush. Using linear or non-linear functions emanating either from the brush center or from the edge of a plateau, the user can distinguish between points that are entirely covered by a brush versus those near the edge of the brush versus those a significant distance from the brush. This is a crucial functionality in multivariate data brushing, as the concept of distance is distributed across all the dimensions and thus difficult to estimate visually.

A frequent use for brushing is *linking* operations in one view to generate corresponding actions in other views. While other forms of linking between subwindows of an application exist, such as when opening a new data file, linked brushing, especially to highlight corresponding data from several views, is probably the most common form of linking found in modern visualization tools. Its popularity stems in a large part from the fact that each view of one's data can reveal interesting features, and by highlighting such a feature in one view, it is possible to build a more complete mental model of the entity by seeing how it appears in other views. This can also help reveal relationships between this entity and others in the data set. For example, when examining multivariate spatial data, it is often useful to jump between the spatially referenced view and the dependent variable view, which often does not preserve the spatial attributes (see Fig. 3).

Another strength of linked brushing is in specifying complex constraints on one's selection. Each type of view is optimized for both conveying certain types



Linking and Brushing. Figure 2. Example of structure-based brushing. Hierarchy view on right shows shape of cluster tree. Area of interest and its level of detail is isolated in red. Corresponding data highlighted in parallel coordinates view on left, with unselected cluster centers shown at a higher level of the hierarchy.



Linking and Brushing. Figure 3. Example of linking spatial and non-spatial views. Diagonal plots are spatial views of remote sensing data, while non-diagonal plots are parameter views. Characteristics of selected region of spatial views are highlighted in other views.

of information as well as specifying conditions on particular types and with a particular degree of accuracy. Thus, for example, one might specify a temporal constraint using a visualization containing a timeline, a constraint on a name field using a sorted list view, and a geographic constraint using a map. While each is effective as a tool for accurate and intuitive specification of a part of a query, none could be used for the complete query.

In some situations the user may want to *unlink* some visualizations in order to maintain a given view while exploring a different area of the data or different data set. Some systems allow the user to indicate for each window whether it is transmitting information to other views and from which other windows it will receive input. A user may also want to constrain the type of information being communicated, as well as its direction. Some types of interaction may be local to a particular window, e.g., zooming in and out, while others are meant to be shared, such as reordering dimensions. Also, in some situations, such as with hierarchically related windows, it may make more sense for the information to move from parent to child, but not the other way. Thus a fairly rich set of connection and communication options may be needed to maximize flexibility.

Key Applications

Linked brushing had its roots in the field of statistics, but is now commonly employed in any field in which visual data analysis is used, including earth and space science, engineering, homeland security, economics, and bioinformatics.

URL to Code

Many commercial and freeware visualization systems support linked brushing. Two such freeware systems are GGobi (http://www.ggobi.org) and XmdvTool (http://davis.wpi.edu/~xmdv).

Cross-references

- ▶ Comparative Visualization
- ▶ Data Visualization
- ▶ Multidimensional Visualization
- ▶ Multivariate Visualization Methods
- ▶ Scientific Visualization
- ▶ Text Visualization
- ▶ Visual Data Mining
- ▶ Visual Interfaces
- ▶ Visualizing Hierarchical Data
- ▶ Visualizing Quantitative Data

Recommended Reading

1. Becker R.A. and Cleveland W.S. Brushing scatterplots. Technometrics, 29(2):127–142, 1987.
2. Becker R.A., Cleveland W.S., and Wilks A.R. The use of brushing and rotation for data analysis. In Dynamic Graphics for Statistics, W.S. Cleveland, M.E. McGill (eds.). Wadsworth, Pacific Grove, CA, USA, 1988, pp. 1–50.
3. Chen H. Compound brushing. In Proc. IEEE Symp. Information Visualization, 2003, pp. 181–188.
4. Cook D. and Swayne D.F. Interactive and Dynamic Graphics for Data Analysis with R and GGobi. Springer, New York, 2008.
5. Doleisch H. and Hauser H. Smooth brushing for focus+context visualization of simulation data in 3D. J. WSCG, 10(1):147–155, 2002.
6. Fua Y.-H., Ward M.O., and Rundensteiner E.A. Structure-based brushes: a mechanism for navigating hierarchically organized data and information spaces. IEEE Trans. Vis. Comput. Graph., 6(2):150–159, 2000.
7. Hauser H., Ledermann F., and Doleisch H. Angular brushing of extended parallel coordinates. In Proc. Information Visualization, 2002, pp. 127–130.
8. Henze C. Feature detection in linked derived spaces. In Proc. Conf. Visualization, 1998, pp. 87–94.
9. Martin A.R. and Ward M.O. High dimensional brushing for interactive exploration of multivariate data. In Proc. IEEE Conf. Visualization '95, 1995, pp. 271–278.
10. McDonald J.A. Orion I: interactive graphics for data analysis. Technical report, Stanford University, 1983.
11. Newton C. Graphica: from alpha to omega in data analysis. In Graphical Representation of Multivariate Data, P. Wang (ed.). Academic, New York, 1978, pp. 59–92.
12. Wills G.J. 524,288 ways to say "this is interesting." In Proc. Information Visualization, 1996, pp. 54–60.
13. Xie Z., Ward M.O., Rundensteiner E.A., and Huang S. Integrating data and quality space interactions in exploratory visualizations. In Proc. Int. Conf. on Coordinated and Multiple Views in Exploratory Visualization, 2007, pp. 47–60.

List

- ▶ Table
- ▶ Text Index Compression

List Comprehension

- ▶ Comprehensions

Literature-based Discovery from Biological Resources

► Text Mining of Biological Resources

Load Balancing

► Database Middleware

Load Balancing in Peer-to-Peer Overlay Networks

Anwitaman Datta
Nanyang Technological University, Singapore, Singapore

Definition

Load balancing in peer-to-peer (P2P) overlay networks is a mechanism to spread various kinds of loads like storage, access and message forwarding among participating peers in order to achieve a fair or optimal utilization of contributed resources such as storage and bandwidth.

Historical Background

Load balancing is a general and critical requirement in distributed and parallel processing systems in order to make efficient and fair use of available resources. In the context of P2P systems, the early works on load-balancing heavily relied on consistent hashing [11], which was proposed in 1997 to originally deal with load-balancing in web caches with minimal movement of data even if new caches are added or if existing ones crash. Consistent hashing was used to achieve storage load-balancing in many early distributed hash table (DHT) P2P networks proposed around 2001.

When a new object is stored, uniform hashing (as used in consistent hashing) helps choosing a peer uniformly from the set of all peers. But the storage load distribution still has high variation as observed in the *balls into bins* phenomenon [14]. The imbalance caused by such variation can be mitigated using an uncoordinated randomized approach called "power of two choices" [13], (More generally, multiple choices.) and was employed for P2P overlays [5] in 2003.

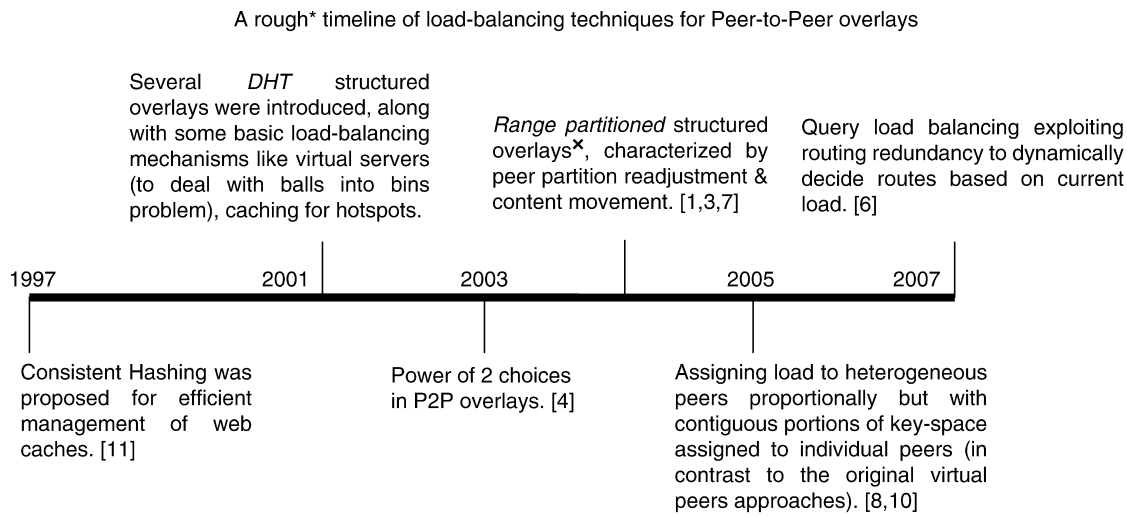
Using uniform hashing leads to loss of existing order relationships such as lexicographic ordering. Objects can be located only based on exact queries. For data-oriented applications involving complex queries like approximate or range queries, uniform hashing is unsuitable. Retaining the ordering relationships however means skewed distribution of the corresponding keys over the key-space of the overlay network. Several new mechanisms and alternate data-structures to dynamically and adaptively partition the key-space based on local load [1,3] and using adaptive reassignment of peers to a different part to the key-space [8] were proposed around 2004–2005 in order to deal with the skewed storage load distribution. Besides workload heterogeneity, the peers themselves have heterogeneous capacities, and assigning proportionally more load to a computer with more resources is another commonly employed approach [4,9] since 2005. Timeline of load-balancing mechanisms developed for P2P overlays is shown in Fig. 1.

While there is an extensive body of work related to storage load-balancing in P2P overlays, it has often been assumed that access load can be dealt with merely by caching. But if each cache is uniformly randomly chosen to answer query for a hot key, it nevertheless suffers from high load imbalance caused by variance, just like the balls into bins problem of uniform hashing. Furthermore, structured overlay routing algorithms do not account for balancing the traffic through each peer, which if unbalanced, may lead to systematic congestion that network level congestion control mechanisms can not deal with. Use of redundant overlay routes to dynamically route requests (e.g., queries) based on load at peers has been proposed in 2007 to deal with traffic and access load balancing [7].

While there have thus been several approaches dealing specifically with diverse kind of load-imbalances, research on a holistic approach (dealing with potentially arbitrary workload skews in terms of storage load, request distribution as well as message forwarding traffic load in the network on one hand, and the capacity of heterogeneous peers particularly in terms of the storage space and bandwidth they have and are willing to contribute) is in an early stage [9].

Foundations

Load in a P2P overlay network is typically either because peers need to store an object, or for answering requests (such as queries) by forwarding them to relevant peers, which may in turn reply back with the



NOTES:
* Some papers with similar ideas were formally published in different calendar years, in which case the timeline indicates the publication of the first paper. The timeline shows the relative order of evolution of ideas, but the distances are not proportional. Also, the citations are not exhaustive.
× The original P-Grid [1]data-structure was designed from its very inception to deal with range-partitioned data, and the first paper was published in 2001, which makes it contemporary with the other DHT papers.

Load Balancing in Peer-to-Peer Overlay Networks. Figure 1. Evolution of load-balancing techniques for P2P overlays over time.

appropriate stored objects. A request may originate at any peer, and the routing algorithm of the overlay is used to forward the request to peers which can respond to it. Thus a request in turn creates both request forwarding and answering loads.

Each of these loads – storage and access load (which in turn leads to request forwarding and answering requests) – have different characteristics, both in terms of the resources they need, as well as the temporal characteristics. Storage load is permanent in that a object need to be stored persistently unless it is removed (deleted) based on application logic. Furthermore, even though storage load distribution changes over time, in general it is expected to change gradually. In contrast, access load is typically temporary, however the load-distribution may change abruptly (e.g., flash crowds). These distinct characteristics provide different opportunities and limitations in load-balancing.

Note that various load-balancing issues may also arise in unstructured and super-peer based overlay networks. These are dealt with heuristically. For instance, super-peers take into account the heterogeneity of peers' resources while designating some peers as super-peers. So the discussion here is primarily restricted to structured overlays.

Approaches for Balancing Storage Load

There have been various approaches to balance storage load among peers. Historically, uniform hashing has been used in many structured overlays as the first step to balance storage load. These are generally called distributed hash tables (DHTs). Subsequent networks overcome the limitations of using uniform hashing. An overlay to store range partitioned data has additional challenges in achieving load-balance. (The term "range-partitioned data" was introduced in 2004 [8] to distinguish structured overlays preserving ordering information from those using uniform hashing (DHTs). The same nomenclature is used here. Though structured overlays supporting range partitioned data came to mainstream focus in 2004–2005 with several new developments [1,3,8], P-Grid [1] supports range-partitioned data but was originally proposed in 2001, making it contemporary to the early DHT proposals.). Next is the discussion of load-balancing issues and mechanisms in both these categories of structured overlays.

Distributed Hash Tables (DHTs)

Hashing. Several pioneering structured overlays used uniform hashing as the primary means to balance storage load. The idea was to hash the object descriptor

to generate a key uniformly (e.g., using cryptographic hashing like SHA) distributed over the key-space. Assignment of the peers to a partition of the key-space was also done uniformly at random, generally using hashing again. This category of structured overlay networks are commonly called distributed hash tables (DHTs). The rationale of the original DHT approach was that if peers were delegated sub-portions of the key-space uniformly randomly, and the keys were distributed uniformly randomly over the key-space, then the peers will have uniform load.

This intuition fails in practice because of what is known as the "balls into bins" effect. If m balls are put in n bins by choosing the bins independently and uniformly randomly for each of the balls, the variation of the number of balls per bin is high [14], and is of the order of m/n where $m >> n$. Since there are usually many more objects than the number of peers (i.e., high m/n), uniform hashing is inadequate to achieve good load-balance. There has been several approaches to reducing the effect of such statistical noise to improve storage load-balancing in DHTs.

Virtual servers (peers). One potential approach to reduce the variance is to have more peers. This can be achieved by creating multiple mutually independent virtual peers per physical computer [6]. Instantiating multiple virtual peers have several down-sides, including the fact that system states (e.g., routing table entries) need to be maintained for each of these virtual peers. Allocating independent key space partitions to more logical (virtual) peers also lead to inefficient search as more overlay level query forwarding are required.

Power of two (multiple) choices. The variance observed in the "balls into bins" process can be reduced without global coordination or knowledge by using an uncoordinated randomized algorithm. Instead of choosing one bin randomly uniformly, several (say two) distinct bins are chosen randomly uniformly as potential destination for a ball. Then the bin with fewer balls is actually chosen to put the current ball in. This mechanism significantly reduces load-imbalance for little extra overhead of choosing multiple (but constant) bins for each ball, instead of picking one [13]. In the context of DHTs, this can be realized by having multiple hash functions to generate multiple possible keys for the same object. The object is actually stored corresponding to that key which belongs to the key-space partition which is stored by the less loaded peer. In order to query an object, all the potential keys for that object need to be queried, thus significantly increasing the query overhead. An alternative is to store pointers at all the other keys to the actual location of the object. This however leads to higher storage and maintenance cost of these pointers.

Virtual peers for heterogeneous physical peers. Use of multiple but equal number of virtual peers was originally proposed to deal with the load-imbalance caused by the balls into bins effect for homogeneous peers [6]. The same idea of virtual peers has also been extended to deal with peer heterogeneity. Proportionally more virtual peers can be created corresponding to a physical node with more resources [10]. To deal with the drawbacks of virtual peers as mentioned earlier, allocation of contiguous partition of the key-space to virtual peers corresponding to the same physical node has been proposed subsequently [4]. Having contiguous key-space allocated to a physical peer has several advantages. There are fewer routing table links to maintain. Also fewer stored objects need to be moved among physically different peers. Note that using such contiguous partitions of the key-space among virtual peers of the same physical node means that there is actually no need of the "virtual peers" abstraction. It is a special case of partitioning the key-space adapted to the granularity of the load and the capacity of the peers as has been used in several other approaches dealing with range partitioned data [3,8,9].

Beyond DHTs

Using uniform hashing to generate keys lead to loss of ordering relationships such as lexicographic ordering, thus uniform hashing based DHTs are ill suited to deal with any queries other than exact queries. DHTs are not suitable for even simple crucial data-oriented queries like approximate queries or range queries. On the other hand, using an order preserving hashing to generate keys leads to potentially arbitrarily skewed and dynamic distribution of keys over the key-space, calling for different load-balancing mechanisms in structured overlays supporting *range partitioned data*. Several overlays exist to support range-partitioned data. While using slightly different data structures, each of these proposals deal with two critical impacts of skewed distributions of keys over the key-space – namely (i) partitioning the key space among peers in a granularity adaptive to the load distribution, and (ii) establishing a data-structure to keep routing

and searching efficient despite non-uniform partitioning of the key-space.

Load-balancing (despite skewed load distribution) leads to allocation of key-space partitions to peers in an uneven manner. This is achieved by re-partitioning key-space of overloaded peers with newly joining peers, or by migrating peers from an under-loaded region. Migration and repartitioning are relatively drastic and need some (partial) global information. The advantage is that it can use any under-loaded peer to alleviate load imbalance for any other peer. In contrast, a localized strategy of readjustment of the key-space partitions with immediate neighbors in the key-space complements the above strategies. By shrinking or expanding the partitions, peers shed (or steal) load by transferring keys to (from) a neighboring peer in the key-space. These strategies are shown in Fig. 2.

In each of these strategies, in absence of global knowledge or coordination, peers need to make autonomous decision about whether it is over-loaded or underloaded with respect to other peers. The load information is obtained by sampling. Often peers exploit their routing topology for sampling a small nevertheless representative subset of the peers. Depending on their local view of the global state, overloaded peers may then request underloaded peers to share load (e.g., as in Mercury [3]), or some underloaded peers may spontaneously decide to relieve overloaded peers (e.g., as in P-Grid [2]). The sampling based load estimate is approximate, moreover the sampling is neither instantaneous nor concurrent over the network, and even if peers had perfect global knowledge, in the absence of coordination, peers make autonomous decisions for load-balancing. All this leads to the risk of oscillatory behavior, which is undesirable. Load-balancing mechanisms do not come free – all of them need transfer of objects, moreover migration also leads to inconsistent routing states at other peers which in turn causes route maintenance overheads. So it is desirable to dampen the load-balancing mechanism somewhat in order to avoid oscillatory effects [2].

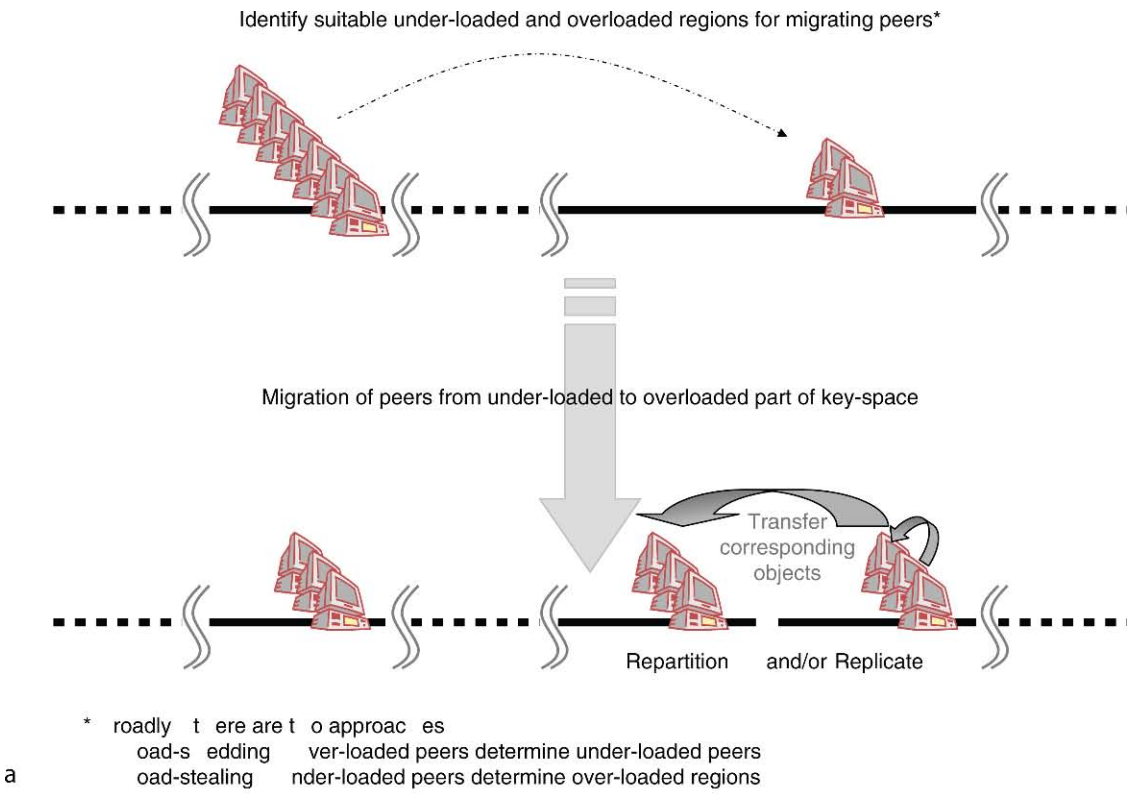
Consequence of load-balancing in overlays supporting range partitioned data. An immediate consequence of load-balancing in structured overlays supporting range partitioned data is that the key-space is partitioned unevenly, and new mechanisms than what is traditionally used in DHTs are required to decide efficient routing table entries and corresponding routing algorithms. Randomized choice of routing tables ensure efficient average routing latency in P-Grid [1] even if the routing tables abstract a (potentially highly) unbalanced trie, while Mercury [3] and Oscar [9] uses randomized routing in a slightly different way, extending the idea of small-world routing [12] over skewed key-space partitions.

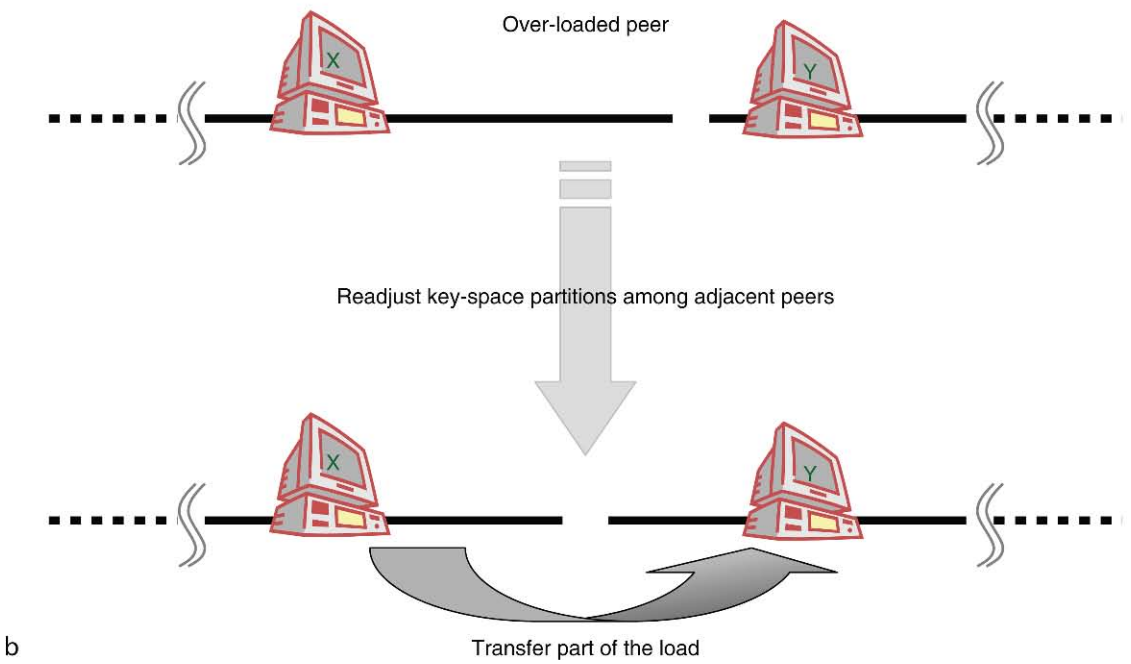
Approaches for Balancing Requests Related Load

Relative popularity of different objects vary, and also change over time. As a consequence, even if storage-load is balanced, peers storing popular objects or services are prone to be overloaded. Replication or caching is the intuitive remedy to deal with hot-spots and dissipate load. If, however, one considers that there are n copies of an object (or service), and there are $m >> n$ requests for the same object, and each request is catered by any one of the copies randomly uniformly, again one encounters a "balls into bins" scenario with high variation in load distribution. Likewise, the load of forwarding messages at different peers have high variation if the routing algorithm of the corresponding overlays are used in a load-agnostic manner. This may in turn lead to congestion caused by the overlay layer, even when there is underutilized resources at other peers. Both these request related load imbalances can be mitigated with the use of simple heuristics exploiting the redundancy of routing choices (which is anyway necessary for fault-tolerance) to the least loaded of the peers that satisfy the routing criterion of the routing algorithm [7]. It has also been shown in the work that either caching or load aware routing is inadequate as stand-alone solutions, but complement each other, and hence need to be used together in order to balance request related loads.

Key Applications

Load-balancing is a critical and desirable property in distributed systems, and is necessary to achieve good performance and make fair and judicious use of available resources. Structured overlays provide indexing mechanism to locate objects distributed over the network with a guaranteed recall based on either exact or range queries, and also support other queries like approximate queries. The basic index can in turn be used to support diverse applications including cooperative file systems, P2P information retrieval, peer data management systems (PDMS) and collaborative work-spaces. For good performance (e.g., response time, fewer failures, etc.) at





Load Balancing in Peer-to-Peer Overlay Networks. Figure 2. Mechanisms for re-balancing load. Note that a minimal redundancy (replication) is always necessary for fault-tolerance, but details of redundancy for fault-tolerance has been omitted for the sake of simplicity. (a) Underloaded peers migrate (new peers joins) to repartition or replicate key-space. (b) Readjust key-space partitions among adjacent peers.

application level, it is imperative that the underlying overlay has good load-balancing.

Future Directions

Peer-to-peer systems have dynamic and skewed workloads, and participating peers have heterogeneous capacities. Consequently, rather than distributing load equally among peers, which is what most current literature aims at, a more pragmatic approach is to look at how to distribute the whole load without violating the autonomous peers' contribution, desirably with minimal wastage of resources. Complementing this, it is also important to find incentive or punishment mechanisms which ensure that peers do contribute resources and do not participate in the system as parasites or free-riders. A holistic design, looking into mechanisms to ensure that autonomous peers contribute sufficient resources, and then allocating these heterogeneous resources to efficiently and effectively cater to dynamic and skewed workloads is the fundamental open problem in the context of load-balancing in peer-to-peer overlays.

Cross-references

- ► Peer Data Management System
- ► Peer to Peer Overlay Networks: Structure, Routing and Maintenance
- ► Routing and Maintenance
- ► Structure

Recommended Reading

1. Aberer K., Datta A., Hauswirth M., and Schmidt R. Indexing data-oriented overlay networks. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
2. Aberer K., Datta A., and Hauswirth M. Multifaceted simultaneous load balancing in DHT-based P2P systems: a new game with old balls and bins. In Self-Properties in Complex Information Systems, Springer, Berlin, 2005.
3. Bharambe A., Agrawal M., and Seshan S. Mercury: supporting scalable multi-attribute range queries. In Symp. on Communications Architectures and Protocols, 2004.
4. Brighten Godfrey P. and Stoica I. Heterogeneity and Load Balance in Distributed Hash Tables. In Proc. 24th Annual Joint Conf. of the IEEE Computer and Communications Societies, 2005.
5. Byers J., Considine J., and Mitzenmacher M. Simple load balancing for distributed hash tables. In Proc. 2nd Int. Workshop on Peer-to-Peer Systems, 2003.
6. Dabek F., Kaashoek F., Karger D., Morris R., and Stoica I. Wide-area cooperative storage with CFS. In Proc. ACM Symp. on Operating Systems Principles, 2001.
7. Datta A., Schmidt R., and Aberer K. Query-load balancing in structured overlays. In Proc. IEEE Int. Symp. on Cluster Computing and the Grid, 2007.
8. Ganesan P., Bawa M., and Garcia-Molina H. Online balancing of range-partitioned data with applications to peer-to-peer systems. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
9. Girdzijauskas S., Datta A., and Aberer K. Oscar: Small-world overlay for realistic key distributions. In Proc. Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2006.
10. Godfrey B., Lakshminarayanan K., Surana S., Karp R., and Stoica I. Load Balancing in Dynamic Structured P2P Systems. In Proc. 23rd Annual Joint Conf. of the IEEE Computer and Communications Societies, 2004.
11. Karger D., Lehman E., Leighton T., Levine M., Lewin D., and Panigrahy R. Consistent hashing and random trees: tools for relieving hot spots on the World Wide Web. In Proc. ACM Symposium on Theory of Computing, 1997.
12. Kleinberg J. The small-world phenomenon: an algorithmic perspective. In Proc. ACM Symp. on Theory of Computing, 2000.
13. Mitzenmacher M. The power of two choices in randomized load balancing. IEEE Trans. Parall. Distrib. Syst., 12 (10):1094–1104, 2001.
14. Raab M. and Steger A. Balls into bins – a simple and tight analysis. In Proc. Int. Workshop on Randomization and Approximation Techniques in Computer Science, 1998.
15. Steinmetz R. and Wehrle K. (eds.). Peer-to-Peer Systems and Applications. Springer Lecture Notes in Computer Science, vol. 3485, 2005. Chapters 9 & 10.

Load Shedding

Nesime Tatbul
ETH Zurich, Zurich, Switzerland

Definition

Data stream management systems may be subject to higher input rates than they can immediately process with their available system resources (e.g., CPU, memory). When input rates exceed the resource capacity, the system becomes overloaded and the query answers are delayed. Load shedding is a technique to remove excess load from the system in order to keep query processing up with the input arrival rates. As a result of load shedding, the system delivers approximate query answers with reduced latency.

Historical Background

Load shedding is a term that originally comes from electric power management, where it refers to the process of intentionally cutting off the electric current on certain lines when the demand for electricity exceeds the available supply, in order to save the electric grid from collapsing. The same term has also been

used in computer networking to refer to a certain form of congestion control approach, where a network router drops packets when its buffers fill up. More recently, load shedding has been proposed as a way to deal with overload in data stream processing systems [4].

Foundations

The goal of load shedding is to make sure that limited system resources operate below their capacity levels in case of unpredictable bursts in data arrival rates. This is achieved by selectively discarding some of the data items, thereby reducing the load at the expense of producing an approximate query answer. The main challenge in this problem is to minimize the loss in answer accuracy.

Assume a set of continuous queries Q, represented as a query plan of operators, where some of these operators may be shared among multiple queries. A set of inputs I feed these queries with streaming data, exerting a total load of $Load(Q(I))$ on a particular system resource with capacity C. A load shedding scheme must address the following key questions:

1. When to shed load? Conceptually, load needs to be shed whenever $Load(Q(I)) > C$.
2. Where to shed load? Load can be discarded at any point in the query plan. Dropping load at earlier points avoids wasting work; however, because of shared operators in the query plan, an early drop might adversely affect the accuracy of too many query answers.
3. How much load to shed? Just enough of the load at the chosen point(s) in the query plan must be shed so that the total resource demand gets below the available capacity with minimal total loss in accuracy.
4. Which data items must be discarded? The data items to be discarded should be chosen based on the approximation model and the properties of the operators in the query plan.

Furthermore, any load shedding scheme must have low run-time overhead in order not to further stress the limited system resources.

Various approaches have been proposed as solutions to the above listed issues. These approaches differ in their assumptions along several dimensions, including:

1. The limited resource under consideration (e.g., CPU, memory, communication bandwidth),
2. The way to reduce load (e.g., drop data, create summaries),
3. The approximation model/objective (e.g., maximum subset, minimum relative error, maximum throughput),
4. The query operator(s) under consideration (e.g., sliding window aggregates, windowed joins),
5. The data arrival model (e.g., stochastic models, temporal models),
6. The control loop (open vs. closed)
7. The system architecture (centralized vs. distributed).

Within the scope of the Aurora Project, Tatbul et al. have proposed a solution framework for load shedding which focuses on CPU as the main scarce resource, and discarding tuples by inserting special drop operators into the running query plan as the load reduction method [14]. The goal in this work is to minimize utility loss in query answers in terms of two alternative QoS (Quality of Service) dimensions: (i) percent tuple delivery, using a *random drop*, or (ii) output values delivered, using a *semantic drop*. A random drop discards tuples based on a drop probability, whereas a semantic drop does so based on a predicate on the tuple content. The earlier the load is reduced in a query plan, the larger is the saving in processing resources. However, shedding load early in a shared query plan may hurt the accuracy for multiple queries. To address this conflict, it is shown that load reduction should be applied either on the input streams, or on streams that immediately follow a shared operator in the query plan. Furthermore, these potential drop locations are ranked in terms of a metric, called *loss/gain ratio*. The drop location that causes the smallest QoS utility loss for the corresponding CPU processing power gained in return per unit drop of data, is preferred over the other drop locations with larger ratios. This way, the overall loss in QoS utility is minimized. For low run-time overhead, this work has proposed to pre-compute a set of load shedding plans based on system statistics, and insatiate these plans at run time based on the observed input rates. The proposed framework has also been extended to handle load shedding on windowed aggregation queries [15]. The key idea is to use a third type of drop operator, called a *window drop*, which semi-probabilistically discards load in units of windows instead of on a per-tuple basis. This way, window integrity can be preserved throughput a query plan, and query answers are guaranteed to be subsets of the original answers. An alternative to the window drop approach was earlier proposed by Babcock

et al. [3]. This work also targets load shedding on aggregation queries under CPU constraints, but uses a different approximation model where the goal is to minimize the maximum relative error across all queries. Drops are applied on a per-tuple basis, leading to query answers with errors in their values. Window statistics and well-known statistical bounds such as the Hoeffding inequality are used to control these errors for a certain set of aggregate functions, including sum, count, and average. A close alternate to dropping tuples under CPU limitations is the selective processing approach proposed by Gedik et al. [8]. This work selectively processes tuples in the stream windows for join operators, in order to maximize the output rate or semantic utility of the query results, in the presence of variations in input rates as well as time correlations between two join inputs.

Load shedding can also be used to deal with memory limitations. Das et al. have focused on this problem for stream joins, where the maximum subset measure is used as the approximation metric [7]. This work assumes a frequency-based data arrival model and proposes two practical heuristics: (i) PROB, which drops tuples from an input stream which had the smallest frequency of occurrence on the opposite stream in the past (assuming that those tuples are the least likely to produce join results also in the future); (ii) LIFE, which drops tuples from an input stream whose product of frequency of occurrence on the opposite stream and remaining window lifetime is the smallest (i.e., the goal is to avoid investing on soon to be expired tuples). Within the scope of the STREAM Project, Srivastava and Widom have proposed an alternative load shedding approach for windowed stream joins in memory-limited environments [12]. This work is based on an age-based data arrival model, where it is assumed that the rate at which a tuple produces join results is solely determined by its age, specified as an age curve. To deal with memory shortage, tuples of a certain age are selectively discarded from the join window to make room for others, which have higher expectation of producing matches. The goal here is again to maximize the size of the join result set. A secondary concern in this work is to be able to produce a random sample from the join in case that the join is followed by an aggregate. In this case, the final output will not be a subset of the exact answer, and the overall goal is then to minimize the relative error, in line with the work of Babcock et al. [3].

Jain et al. have proposed a load shedding approach to reduce the network bandwidth usage [9]. This approach is based on Kalman Filters which can be used to model data streams as processes with states that evolve over time. As new tuples arrive at a source site, it is checked if the current model installed at the remote server site can still answer the query within given precision bounds. If so, there is no need to send the new tuple to the server, i.e., it can be discarded. Otherwise, the server model has to be updated, hence the new tuple is transmitted. Adaptivity is achieved by adjusting model parameters to changing load characteristics.

Stream load can also be reduced based on creating summaries of data instead of discarding data. This idea was pursued by two different lines of work within the scope of the TelegraphCQ Project: (i) Reiss and Hellerstein have proposed a load shedding technique called *data triage*, where excess data is not dropped, but stored in synopsis data structures [11]. At the end of a well-defined query window, the stored synopses are processed through a shadow query plan to compute an approximate result on the summarized portion of the data. Finally, exact and approximate results are merged into one composite result for that query window. This works using an error model based on Minkowski distance. (ii) Chandrasekaran and Franklin have focused on hybrid queries that process live data streams in correlation with historical data archived on disk [5]. In this case, disk becomes the bottleneck resource. To keep processing of disk data up with processing of live data, disk data is organized into multiple resolutions of reduced summaries. Depending on the live data rates, the system picks the right resolution summary to use in query processing. This is a form of load shedding that tries to cut down from disk access cost using data summaries.

The NiagaraCQ Project has taken an integrated approach where load shedding is seen as an extension to continuous query optimization. Kang et al. use a unit-time-based cost model where total cost of join processing is broken into two components, one for each join direction [10]. The optimal index and join algorithm combination for each direction is determined so as to maximize query throughput. Under CPU and memory limitations, the optimizer determines the ideal rate for each input and accordingly places a random drop to control the input rates. Ayad and Naughton use a similar analytical cost model, but

extend it to plans with multiple joins [2]. It is shown that if computational resources are enough, then all join plans have the same throughput, however, they may substantially differ in their resource utilization. If all of these plans are infeasible (i.e., lead to CPU overload), then load must be shed via random drops. The focus is on picking the right join plan, the locations on the plan to insert the drops, and the amount of drops. An interesting result shown in this work is that the optimal join plan (i.e., with the lowest utilization) when resources are sufficient is not necessarily the optimal plan (i.e., with the highest throughput) when resources are insufficient.

All of the above described approaches assume that stream processing is performed on a single server. The overload problem can also arise in distributed stream processing systems where queries are distributed onto multiple servers. In a distributed environment, there is load dependency among the nodes that are assigned to run pieces of the same query. As a result, shedding load at an upstream node affects the load levels at its downstream nodes, and the load shedding actions at all nodes along a query plan will collectively determine the quality degradation at the query end-points. Within the scope of the Borealis Project, Tatbul et al. have modeled this problem as a linear optimization problem, and proposed two alternative solutions: (i) a centralized approach, where a coordinator node produces globally optimal plans with the help of an LP solver, and the rest of the nodes adopt their share of these global plans; (ii) a distributed approach, where nodes exchange metadata information (represented in the form of a feasible input table (FIT) which shows input rate combinations that are feasible for a given node) with their neighbors, and each node produces its own plan based on the available metadata [13]. Both of these solutions are based on the idea of pre-computing the load shedding plans in advance and storing them in a quadtree-based plan index. It is shown that the FIT-based plan generation is more efficient than its solver-based counterpart. Furthermore, the distributed solution is expected to be more responsive in dynamic environments due to its ability to incrementally update previously computed load shedding plans, reducing the amount of run-time communication needed among the nodes.

These approaches are all open-loop solutions in that the system load is periodically monitored and the load shedding algorithms are triggered as necessary. There has also been recent work that applies control-theoretic concepts to finer-grained adaptive load shedding on data streams [1,16]. These approaches are based on constructing a feedback loop that continually monitors the high-frequency variations in system parameters and makes the necessary adjustments in the load controllers accordingly. Such closed-loop approaches are shown to be more adaptive for input workloads with higher frequency fluctuations in stream data rates.

Load shedding finds use also in resource-intensive data stream mining applications. As argued by Chi et al. [6], in common data mining tasks such as classification and clustering of multiple data streams, the impact of load shedding on performance is not known a priori as the mining quality often depends on specific feature values observed in the stream in a non-monotonic way. This requires feature value prediction and adaptation. The Loadstar scheme uses a Markov model to predict the distribution of future feature values whose parameters are adaptively updated in time in order to maximize the classification quality under CPU constraints [6]. The high-level idea in this work is to allocate more resources to data streams that carry more uncertainty while shedding the ones whose class labels are more certain for the upcoming time window.

Key Applications

Load shedding can be used in all data-intensive streaming applications for which low latency answers can be more critical than full answer accuracy. These include sensor-based monitoring (e.g., habitat monitoring, bio-medical monitoring, weather monitoring, road traffic monitoring), RFID-based asset tracking, GPS-based location tracking, video-based security monitoring, and network traffic monitoring.

Future Directions

Load shedding in data stream management systems is currently an active area of research. A significant body of research results has been produced in this area since circa 2002. The future directions include development of new load shedding schemes for other sets of assumptions along the dimensions listed above. There is also a need to integrate the complementary and alternative solution schemes under a single framework, which could automatically select the right set of techniques for a broad range of system resources, based on the characteristics of the received workload as well as the application-specific quality of service criteria.

Cross-references

- ▶ Adaptive Stream Processing
- ▶ Approximate Query Processing
- ▶ Data Stream Management Architectures and Prototypes
- ▶ Continuous Query
- ▶ Data Sampling
- ▶ Data Sketch/Synopsis
- ▶ Data Stream
- ▶ Data Quality Dimensions
- ▶ Data Quality Models
- ▶ Data Reduction
- ▶ Stream Mining
- ▶ Stream-Oriented Query Languages and Operators
- ▶ Stream Processing
- ▶ Stream Sampling
- ▶ Wavelets on Streams
- ▶ Window-Based Query Processing
- ▶ Windows

Recommended Reading

1. Amini L., Jain N., Sehgal A., Silber J., and Verscheure O. Adaptive Control of Extreme-scale Stream Processing Systems. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006.
2. Ayad A. and Naughton J.F. Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
3. Babcock B., Datar M., and Motwani R. Load Shedding for Aggregation Queries over Data Streams. In Proc. 20th Int. Conf. on Data Engineering, 2004.
4. Carney D., Çetintemel U., Cherniack M., Convey C., Lee S., Seidman G., Stonebraker M., Tatbul N., and Zdonik S. Monitoring Streams - A New Class of Data Management Applications. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
5. Chandrasekaran S. and Franklin M.J. Remembrance of Streams Past: Overload-Sensitive Management of Archived Streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
6. Chi Y., Yu P.S., Wang H., and Muntz R.R. Loadstar: A Load Shedding Scheme for Classifying Data Streams. In Proc. SIAM International Conference on Data Mining, 2005.
7. Das A., Gehrke J., and Riedewald M. Approximate Join Processing Over Data Streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
8. Gedik B., Wu K., Yu P.S., and Liu L. CPU Load Shedding for Binary Stream Joins. Knowl. and Inf. Syst., 13(3):271–303, 2006.
9. Jain A., Chang E.Y., and Wang Y. Adaptive Stream Resource Management using Kalman Filters. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
10. Kang J., Naughton J.F., and Viglas S. Evaluating Window Joins over Unbounded Streams. In Proc. 19th Int. Conf. on Data Engineering, 2003.
11. Reiss F. and Hellerstein J.M. Data Triage: An Adaptive Architecture for Load Shedding In TelegraphCQ. In Proc. 19th Int. Conf. on Data Engineering, 2005.
12. Srivastava U. and Widom J. Memory Limited Execution of Windowed Stream Joins. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
13. Tatbul N., Çetintemel U., and Zdonik S. Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
14. Tatbul N., Çetintemel U., Zdonik S., Cherniack M., and Stonebraker M. Load Shedding in a Data Stream Manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003.
15. Tatbul N. and Zdonik S. Window-aware Load Shedding for Aggregation Queries over Data Streams. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.
16. Tu Y., Liu S., Prabhakar S., and Yao B. Load Shedding in Stream Databases: A Control-Based Approach. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.

LOC METS

Prasenjit Mitra
The Pennsylvania State University, University Park, PA, USA

Synonyms

Metadata encoding and transmission standard; Library of congress METS

Definition

The Library of Congress (LOC) Metadata Encoding and Transmission Standard (METS) is an XML (Extensible Markup Language) based format used for encoding metadata. The metadata is used to markup digital library objects in a repository or for exchange across repositories. METS is a Digital Library Federation initiative that is a successor to the Making of America II project (MOA2).

Key Points

The MOA2 project attempted to provide encoding formats for descriptive, administrative, and structural metadata for text and image-based documents. (http://www.loc.gov/standards/mets/METSOverview.v2.html) The Digital Library Federation (DLF) sponsored the project and the National Endowment for the Humanities funded it. MOA2 involved discussions led by the University of California, Berkeley with participants from New York Public Library and the libraries of Cornell, Penn State, and Stanford universities. The project produced a Document Type Definition

(DTD) that specifies a vocabulary and syntax for encoding digital objects. (http://www.lib.berkeley.edu/digicoll/bestpractices/mets_history.html) The library community realized that the MOA2 DTD was too restrictive, because, MOA2 did not provide some basic functionality required for multimedia objects like video and audio. METS arose from efforts to address these problems in MOA2.

Digital objects and the metadata needed to describe them are different from the metadata required for documents. Digital objects require structural metadata that indicates how the components of the object are glued together and technical metadata that specifies how the digital object was produced. For example, if a digital object contains image and text files, the structural metadata indicates the hierarchical structure of these objects and files. Furthermore, without these metadata, the authenticity of a digital object could be in doubt. METs allows specifications of the structural technical metadata, as well as metadata required for internal management and administration.

A METS document consists of seven sections:

- *METS header*: describes the document itself and publishes information about the creator, editor, etc.
- *Descriptive metadata*: both external, i.e., residing outside the document and internal.
- *Administrative metadata*: describes how the object was created and stored, specifies intellectual property rights, etc. Like the descriptive metadata, administrative metadata can also be both external or internal.
- *File Section*: lists all files that comprise the digital object. Groups of files can be specified using <fileGrp> with individual file elements specified using <file>.
- *Structural Map*: specifies the hierarchical structure of the digital object.
- *Structural Links*: allows creators to link different nodes in the hierarchy using hyperlinks.
- *Behavioral*: associates executable behaviors with content in the METS object.

METS is specified as an XML Schema and can be used in the role of a Submission Information Package (SIP), Archival Information Package (AIP) or Dissemination Information Package (DIP) within the Open Archival Information System (OAIS) Reference Model (http://ssdoo.gsfc.nasa.gov/nost/isoas/ref_model.html) [1–3].

Cross-references

▶ Digital Libraries

Recommended Reading

1. Gartner R. METS: Metadata Encoding and Transmission Standard. JISC Techwatch Report, 2002.
2. Guenther R. and McCallum S. New Metadata Standards for Digital Resources: MODS and METS. Bulletin of the American Society for Information Science and Technology, 2003.
3. Cundiff M.V. An introduction to the Metadata Encoding and Transmission Standard (METS). Library Hi Tech, 2004.

Local Web Search

▶ Geo-Targeted Web Search

Localization Abstraction

▶ Abstraction

Locality

▶ Locality of Queries

Locality of Queries

Pablo Barceló
University of Chile, Santiago, Chile

Synonyms

Locality; Hanf-locality; Gaifman-locality

Definition

Let σ be a relational signature without constant symbols. Given a σ-structure $\mathcal{A}$, its *Gaifman graph*, denoted by $G(\mathcal{A})$, has A (the domain of $\mathcal{A}$) as the set of nodes. There is an edge (a_1, a_2) in $G(\mathcal{A})$ iff there is a relation symbol R in σ such that for some tuple t in the interpretation of this relation in $\mathcal{A}$, both a_1, a_2 occur in t. The *distance* $d(a_1, a_2)$ is the distance in the Gaifman graph, with $d(a, a) = 0$. If $\bar{a}$ and $\bar{b}$ are tuples of elements, then $d(\bar{a}, \bar{b})$ stands for the minimum of $d(a, b)$, where $a \in \bar{a}$ and $b \in \bar{b}$.

Let $\mathcal{A}$ be a σ-structure, and $\bar{a} = (a_1,...,a_m) \in A^m$. The *radius r ball around* $\bar{a}$ is the set $B_r^{\mathcal{A}}(\bar{a}) = \{b \in A \mid d(\bar{a}, b) \leq r\}$. The *r-neighborhood of* $\bar{a}$ *in* $\mathcal{A}$ is the structure $N_r^{\mathcal{A}}(\bar{a})$ over signature σ expanded with m constant symbols, where the universe is $B_r^{\mathcal{A}}(\bar{a})$, the σ-relations are restrictions of the σ-relations in $\mathcal{A}$ to $B_r^{\mathcal{A}}(\bar{a})$, and the m additional constants are interpreted as $a_1,...,a_m$. Notice that any isomorphism h between $N_r^{\mathcal{A}}(a_1, ...,a_m)$ and $N_r^{\mathcal{B}}(b_1,...,b_m)$ imposes that $h(a_i) = b_i$, $1 \leq i \leq m$.

Let $\mathcal{A}, \mathcal{B}$ be σ-structures, $\bar{a} \in A^m$ and $\bar{b} \in B^m$. Then $(\mathcal{A}, \bar{a}) \rightleftarrows_r (\mathcal{B}, \bar{b})$ if there exists a bijection $f : A \to B$ such that $N_r^{\mathcal{A}}(\bar{a}c) \cong N_r^{\mathcal{B}}(\bar{b}f(c))$, for every $c \in A$.

Hanf-locality. An m-ary query Q on σ-structures is *Hanf-local*, if there exists $r \geq 0$ such that for every σ-structures $\mathcal{A}$ and $\mathcal{B}$, and every $\bar{a} \in A^m$ and $\bar{b} \in B^m$, if $(\mathcal{A}, \bar{a}) \rightleftarrows_r (\mathcal{B}, \bar{b})$ then $(\bar{a} \in Q(\mathcal{A}) \iff \bar{b} \in Q(\mathcal{B}))$.

Gaifman-locality. An m-ary query Q, $m > 0$, on σ-structures, is *Gaifman-local* if there exists $r \geq 0$ such that for every σ-structure $\mathcal{A}$, and every $\bar{a}, \bar{b} \in A^m$, if $N_r^{\mathcal{A}}(\bar{a}) \cong N_r^{\mathcal{A}}(\bar{b})$ then $(\bar{a} \in Q(\mathcal{A}) \iff \bar{b} \in Q(\mathcal{A}))$.

Key Points

Locality is a property of queries that finds its origins in the work by Hanf [1] and Gaifman [2], and that has shown to be very useful in the contexts of finite model theory and relational database theory. In very rough terms, a query is local if its truth value only depends on a small neighborhood of the input around its free variables. Locality is primarily used to prove inexpressibility results over finite structures, but it can also be used for establishing normal forms for logical formulae. It is a particularly helpful tool for finding easy winning strategies for the duplicator in the Ehrenfeucht-Fraïssé game, avoiding complicated combinatorial arguments.

The abstract study of the concepts behind the locality theorems of Hanf and Gaifman was initiated in [3]. It is shown there that the two notions of locality presented above are related: Every m-ary query (with $m > 0$) that is Hanf-local is also Gaifman-local, but the converse does not hold. The most paradigmatic example of a query that is neither Hanf- nor Gaifman-local is the one that computes the transitive closure of a graph.

A typical application of locality to prove the inexpressibility of query Q over logic $\mathcal{L}$ is done in two steps. First, show that Q is not local, and second, prove that every query defined by a formula in $\mathcal{L}$ is local. It follows from [3,4] that first-order logic, as well as many of its extensions with counting and generalized quantifiers, only define queries that are both Hanf- and Gaifman-local. This gives a simple proof of the fact that none of these logics can compute the transitive closure of a graph.

Cross-references

▶ Ehrenfeucht-Fraïssé Games

Recommended Reading

1. Hanf W. Model-theoretic methods in the study of elementary logic. In J.W. Addison et al. (eds.). The Theory of Models. North Holland, Amsterdam, 1965, pp. 132–145.
2. Gaifman H. On local and non-local properties. In Proc. of the Herbrand Symp., Logic Colloquium '81, North Holland, Amsterdam, 1982.
3. Libkin L. On the forms of locality over finite models. In Proc. 12th Annu. IEEE Symp. on Logic in Computer Science, 1997, pp. 204–215.
4. Libkin L. On counting logics and local properties. ACM Trans. Computat. Logic 1(1):33–59, 2000.

Locality of Reference

▶ Memory Locality

Locality Principle

▶ Memory Locality

Locality-Preserving Mapping

▶ Space Filling Curves
▶ Space-Filling Curves for Query Processing

Location Prediction

▶ Spatial Data Mining

Location Services

▶ Location-Based Services

Location-Based Services

Scott A. Bridwell, Harvey J. Miller
University of Utah, Salt Lake City, UT, USA

Synonyms

Location services; Geographic information services; Mobile map services; LBS

Definition

Location-based services (LBS) provide targeted information to individuals based on their geographic location in real or near-real time, typically through wireless communication networks and clients such as portable computers, personal digital assistants, mobile phones, and in-vehicle navigation systems.

Historical Background

LBS have emerged from the convergence of three major technological trends: (i) *geospatial technologies*, including location-aware technologies, geographic information systems (GIS) and spatial databases; (ii) *the Internet*, and; (iii) *information and communication technologies*, in particular, personal computing devices and mobile communication. Some of these technologies date to the late 1960s and early 1970s. For example, the first GIS was developed in the mid-1960s, while the global positioning system, the Integrated Services Digital Network (ISDN), mobile telephony and TCP/IP as a dominant network protocol emerged in the 1970s. The personal computer was introduced in early 1980s, an era that also experienced the deregulation of the telecommunications industry in the USA and Europe. Through the 1990s and early twenty-first century, these technologies matured, diffused and converged sufficiently such that LBS in the contemporary sense is possible [4,8].

Foundations

Location-Aware Technologies

Location-aware technologies (LATs) are devices that can report their geographic location in real or near-real time. Technologies for determining geographic location include the global positioning system, radiolocation methods and interpolation [2]. These can be used in combination.

The *global positioning system* (GPS) exploits time differences of signals arriving from subset of a satellite constellation in Earth orbit. The GPS is traditionally the most common LAT due to its high accuracy and low cost. GPS receivers are becoming small and light enough to embed in many other mobile technologies. However, the GPS requires line-of-sight with orbital satellites; this can be a problem in places with dense foliage or tall buildings, as well as inside built structures.

Radiolocation methods exploit wireless communication systems and determine location using methods such as those based on the time, time difference or angle of the signals' arrivals at base stations from mobile clients. The configuration of the network determines the precision of this method, with greater precision for areas with higher population densities and larger numbers of base stations. Radiolocation methods are less accurate in rural areas and have caused difficulties for wireless carriers attempting to achieve the E911 standards.

Other radiolocation methods include Bluetooth, WiFi, and Radio Frequency Identification (RFID) tags. These methods utilize a fixed network of sensory objects that sense objects in close proximity, interact with other sensors to triangulate the locations of an object based on some measure of signal strength, or utilize training information containing combinations of spatial coordinates and signal strengths. However, these methods are expensive and only cover small coverage areas. Their integration with positioning mechanisms outside of the building may also prove difficult.

Interpolation methods use distances and directions along a route from a known location to determine the current location.

Locational and Spatial Data Management

Since LBS users are likely to be mobile, a fundamental issue concerns the process whereby user locations are communicated to the central database and location server that supports LBS queries. At the time of the query the location service must have an accurate estimate of the user's location. The strategies used to communicate this information vary depending on the capabilities of the mobile device, the required accuracy for a particular query and the costs levied against the application server. This is essentially a question of how often the location of a moving object should be updated in the database.

Simple update strategies update a user's location whenever their location changes. This assumes there is no uncertainty in the tracking process and places high communication burdens on the servers. Temporal update strategies provide periodic updates of a user's

location based on a recurring time interval. In the interval between updates, a user's location must be estimated according to an interpolation function. The accuracy of this estimate is based on the duration of the temporal recurrence interval, the distance the user travels between samples and assumed maximum travel speed. Distance or spatial update strategies provide updates when the user has moved a specified distance. This has the advantage of being more sensitive to the behavior of the user than temporal strategies. Dead-reckoning strategies integrate the temporal and distance approaches by comparing an estimated location with a measured location; this usually occurs on the client device. Updates to the database or location server occur when the difference between the estimated and measured location exceeds a given threshold.

Additional spatial data management issues include representations of geographic space and georeferenced content of real-world services such as restaurants and shops. While the former often includes physical features such as mountains, coastlines, and so forth, of critical importance is the transportation network. Transportation networks are important since they serve as the basis for navigation as well as the basis for georeferencing through street addresses. Required are multiple and integrated network representations that can support both navigation and locational referencing. The latter includes both the user's position provided by the LAT as well as linear referencing within the network [3].

Middleware, Open Standards and Interoperability

LBS middleware provides standard mechanisms for connecting the software components necessary for a given service. These components may be entirely present within a single device or distributed across a network of location, data and application servers. The primary objective of middleware is to support the interoperability of applications across devices and mobile service networks with different locational positioning methods and protocols. Middleware also provides a generic interface for querying geographic information stored in different formats and database systems.

The *Java ME Location API* is an example of device-oriented middleware. The API provides an interface defining methods for querying the location of the user with information corresponding to the confidence or uncertainty of the estimated location. This application resides on the device.

The *Open Location Services* (OpenLS) initiative is an example of set of standards to support network-oriented middleware. The OpenLS defines a set of core services and functions that are expected to satisfy most LBS applications. These services are similar database query languages in that they define an explicit structure for asking questions and interpreting the answers. Queries and results are structured using the eXtensible Markup Language (XML). Gateway services provide the mechanisms for querying the location of the user or other users required for the given application (e.g., friend finding); applications utilizing this service may exist on the device or be accessed through the network. Locational utility services provide the means for translating street addresses to geographical coordinates and vice versa. Directory services allow for querying points of interest (POI) such as restaurants. Route services provide a route, between two locations according to the preferences of the user. Presentation services allow for displaying querying results on a graphical map. Each of these services may exist on different providers. Using this framework, an LBS application developer could chain these services together in a similar manner to importing a software library or querying a relational database [7].

Locational Privacy

Detailed movement patterns in space and time are a signature that reveals much about an individual. Locational privacy is an emerging concept that suggests individuals have rights to their signature in space and time and can determine when, how and to what extent location information is communicated to others. Strategies for protecting locational privacy include regulation, privacy policies, anonymity and obfuscation. Regulation and privacy policies are trust-based mechanisms for defining unacceptable uses of location information. However, trust can be broken, making these strategies vulnerable to unintentional and intentional disclosure. Anonymity detaches locational information from an individual's identity. However, GIS can integrate locational information with other data such as remotely sensed imagery, geo-referenced social, economic and cadastral data, point-of-sale data, credit card transactions, traffic monitoring and video surveillance imagery, and other geosensor network data, allowing identity to be inferred. Obfuscation techniques deliberately degrade locational information, using error and uncertainty to protect privacy. Obfuscation techniques include geographic masking for static and mobile data (see [1]).

Location-Based Services. Table 1. Common LBS applications

• News	• Emergency response
• Navigation	• Person finding (friends, children)
• Traffic information	• Pet tracking
• Points of interest	• Electronic toll collection
• Advertising	• Car tracking
• Gaming	• Fleet management
• Weather forecasts	• Asset management

Key Applications

Key LBS applications span a large spectrum from convenient, "concierge" services to critical emergency response. Table 1 lists common LBS applications [9].

Two possible dimensions for classifying LBS applications are person versus device-oriented and push versus pull services [9]. *Person-oriented LBS* encompasses applications that are user-based. The user typically has control of the service: these intend to locate the person and/or use that location to enhance a service. These applications include news, navigation, points of interest, traffic, emergency response, and so on. *Device-oriented LBS* are not controlled by a person. Rather, the intent is to track an object or a set of objects. Applications include asset and fleet management.

Push services are LBS where a person receives information as a result of his or her location without actively requesting it. This information could be sent to the user based on prior consent (e.g., weather or traffic warnings) or without consent (e.g., advertising). *Pull services* are those where the user activity requests the information (e.g., points of interest, navigation, weather forecasts, traffic information).

Cross-references

► Mobile Objects Databases

Recommended Reading

1. Duckham M., Kulik L., and Birtley A. A spatio-temporal model of strategies and counter-strategies for locational privacy protection. In Geographic Information Science – Proc. 4th International Conference, 2006, pp. 47–64.
2. Grejner-Brzezinska D. Positioning and tracking approaches and technologies. In Telegeoinformatics: Location-based Computing and Services, H.A. Karimi, A. Hammad (eds.). CRC Press, Boca-Raton, FL, 2004, pp. 69–110.
3. Jensen C. Database aspects of location-based services. In Location-Based Services, J. Schiller, A. Voisard (eds.). Morgan Kaufmann, New York, 2002, pp. 115–145.
4. Jiang B. and Yao X. Location-based services and GIS in perspective. Comput. Environ. Urban. Syst., 30:712–725, 2006.
5. Kolodziej K.W. and Hjelm J. Local Positioning Systems: LBS Applications and Services. Taylor and Francis, London, 2006.
6. Küpper A. Location-Based Services: Fundamentals and Operation. Wiley, Hoboken, NJ, 2005.
7. Lopez X.R. Location-based services. In 1Telegeoinformatics: Location-based Computing and Services, H.A. Karimi, A. Hammad (eds.). CRC Press, Boca Raton, FL, 2004, pp. 171–188.
8. Shiode N., Li C., Batty M., Longley P., and Maguire D. The impact and penetration of location-based services. In Telegeoinformatics: Location-based Computing and Services, H.A., Karimi A. Hammad (eds.). CRC Press, Boca Raton, FL, 2004, pp. 349–366.
9. Spiekermann S. General aspects of location-based services. In Location-Based Services, J. Schiller, A. Voisard (eds.). Morgan Kaufmann, New York, 2004, pp. 9–26.

Lock Coupling

► B-Tree Locking

Lock Manager

► Concurrency Control Manager

Lock Tuning

► Tuning Concurrency Control

Locking Granularity and Lock Types

Ralf Schenkel
Max-Planck Institute for Informatics, Saarbrücken, Germany

Synonyms

Locking granularity and lock types

Definition

Databases are usually organized hierarchically, with tablespaces containing tables, which in turn contain

records. In multigranularity locking, this organization is exploited for a more efficient lock management by allowing transactions to lock objects of different granularities like tables or records. Thus, instead of locking each record of a table separately, a transaction can lock the complete table. To ensure a correct execution when transactions use different granularities for locking, additional lock modes are introduced to avoid nonserializable executions.

Locking Granularity and Lock Types. Table 1. Compatibility matrix of lock types

	S	X	IS	IX	SIX
S	+	–	+	–	–
X	–	–	–	–	–
IS	+	–	+	+	+
IX	–	–	+	+	–
SIX	–	–	+	–	–

Key Points

Transactions that acquire many locks on small items like records or pages incur a non-negligible performance and memory overhead for managing these locks. Such transactions can benefit from acquiring locks on coarser granules like tables or complete tablespaces, avoiding many fine-grained locks. However, concurrency may be lower due to an increased number of conflicts with coarser locks. On the other hand, for transactions that access only a few records, locking these records directly will usually be the best solution.

Concurrent transactions that use different granularities for locking cannot easily coexist if any guarantees on the serializability of the execution should be provided, as locks of different granularities do not conflict with each other and hence cannot prevent any nonserializable executions. In such a *multi-granularity locking scheme*, transactions wanting to acquire a lock on a smaller granule must first acquire locks on all larger granules. While it would be sufficient if the locks on the coarser granules were acquired in the same mode as the lock on the smaller granule, this would lead to many unnecessary blockings due to conflicts on the coarser granules. As an example, consider two concurrent transactions that want to modify different records of the same table and hence need to acquire an exclusive lock on these records. As they modify different records, they do not conflict with each other. However, they would additionally have to acquire an exclusive lock on the table, too, which would make the second transaction wait until the first released the lock again.

Such performance penalties can be circumvented with additional lock types for coarser granules that express the *intended* lock type the transaction wants to acquire on the smaller granule: The lock modes *IS (intentional shared)* and *IX (intentional exclusive)* on the coarse granules correspond to *S (shared)* and *X (exclusive)* on the smaller granule. These lock types coexist with the standard shared and exclusive lock types on the coarse granule (that are used to get shared or exclusive access to *all* items of the smaller granule).

The additional lock type *SIX (shared intentional exclusive)* combines an S and an IX lock. It is used when the transaction plans to read most of the items within the granule and additionally modify some of them. In this situation, an X lock would be too restrictive as it would lock out any other transaction from reading items in that table.

To ensure serializability in a system with multigranularity locking, a transaction wanting to acquire a lock on any granule (for example, on a record of a table) must first acquire *warning locks* with the corresponding intentional lock type on all coarser granules. Table 1 shows the compatibility matrix for the different lock types.

The best locking granularity for a transaction may change throughout its execution. For example, if the transaction initially plans to access only a few records, it may be best to lock only records; if the access pattern changes later and it turns out that it is necessary to access all or almost all records of the table, it may be better to *escalate* the record-level locks to a single lock on the table. To do this, the transaction must first convert the warning lock (of type IS or IX) on the table to a "real" lock (of type S or X), for which it may have to wait until other transactions have released incompatible locks. It can release any record-level locks subsumed by the new lock on the table without compromising serializability.

Cross-references

- ▶ B-Tree Locking
- ▶ Locking Granularity and Lock Types
- ▶ Serializability
- ▶ SQL Isolation Levels
- ▶ Two-Phase Locking

Recommended Reading

1. Jim Gray, Raymond A. Lorie, Gianfranco R. Putzolu, and Irving L. Traiger. Granularity of locks in a large shared data base. In Proc. 1st Int. Conf. on Very Large Data Bases, 1975, pp. 428–451.
2. Gerhard Weikum, and Gottfried Vossen. Transactional Information Systems. Morgan Kaufman, San Francisco, CA, 2002.

Locking Protocol

► Two-Phase Locking

Log Component

► Logging/Recovery Subsystem

Log Manager

► Logging/Recovery Subsystem

Logging and Recovery

Erhard Rahm
University of Leipzig, Leipzig, Germany

Synonyms

Failure handling; Rollback; Undo; Redo; Checkpoint; Backup; Dump

Definition

Logging and recovery ensure that failures are masked to the users of transaction-based data management systems by providing automatic treatment for different kinds of failures, such as transaction failures, system failures, media failures and disasters. The main goal is to guarantee the atomicity (A) and durability (D) properties of ACID transactions by providing undo recovery for failed transactions and redo recovery for committed transactions. Logging is the task of collecting redundant data needed for recovery.

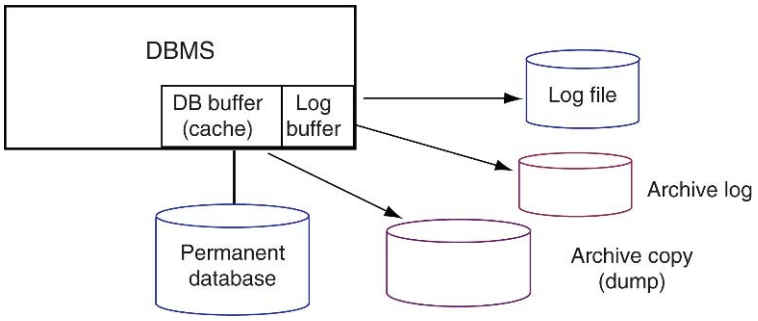
Key Points

The ACID concept requires that no data changes of failed transactions remain in the database. Failed transactions thus have to be rolled back by undoing all their changes (undo recovery). On the other hand, data changes of successfully ended (committed) transactions must not be lost but have to survive possible failures. Failure treatment thus implies a redo recovery for committed transactions. Recovery support is typically provided for transaction failures during normal processing (transaction recovery), for system failures (system recovery), and media failures (media recovery). System and media recovery are also known as two kinds of "crash recovery." In addition, disaster recovery can deal with the complete destruction of a computer center, e.g., due to an earthquake or terror attack. Recovery is typically based on logging, i.e., the collection of protocol data recording which transactions have been executed and which changes have been performed by them.

Figure 1 shows components of a central database management system (DBMS) involved in logging and recovery. The database objects (e.g., tables, records) are persistently stored in the *permanent database*, typically on one or several disks. All database operations including updates are performed in main memory. For this reason, pages of the database are cached in a main memory buffer (database buffer). Log records are persistently stored in a sequential *log file* on dedicated disks. Log records are written for the start, rollback and commit of transactions as well for every database change. For performance reasons log records are first collected in a log buffer in main memory, which is written to the log file when it becomes full or when a transaction commits. A transaction is committed when its commit record is logged on the log file.

Numerous approaches have been proposed and current database management systems provide efficient implementations for logging and recovery. The major tasks to be solved for dealing with the mentioned types of failures are:

- *Transaction recovery* (rollback) is performed when a transaction fails during normal processing, e.g., due to a program error or invalid input data. The log records in the log buffer and in the log file are used to undo the changes of the failed transaction in reverse order.
- *System (crash) recovery* is needed when the whole database (transaction) system fails, e.g., due to a hardware or software error. All transactions which were active and not yet committed at crash time



Logging and Recovery. Figure 1. DBMS components involved in logging and recovery.

have failed so that their changes must be undone. The changes for transactions that have committed before the crash must survive. A redo recovery is needed for all changes of committed transactions that have been lost by the crash because the changed pages resided only in main memory but were not yet written out to the permanent database. Periodically writing out modified pages, e.g., within so-called *checkpoints*, help to reduce the amount of redo work during crash recovery. Furthermore, the number of relevant log records and thus the size of the log file can be reduced by checkpoints.

- *Media (crash) recovery* deals with failures of the storage media holding the permanent database, in particular disk failures. The traditional database approach for media recovery uses archive copies (dumps) of the database as well as archive logs (see Fig. 1). Archive copies represent snapshots of the database and are periodically taken. The archive log contains the log records for all committed changes which are not yet reflected in the archive copy. In the event of a media failure, the current database can be reconstructed by using the latest archive copy and redoing all changes in chronological order from the archive log. A faster recovery from disk failures is supported by disk organizations like RAID (redundant arrays of independent disks) which store data redundantly on several disks. However, they do not eliminate the need for archive-based media recovery since they cannot completely rule out the possibility of data loss, e.g., when multiple disks fail.
- *Disaster recovery* can be achieved by maintaining a backup copy of the database at a geographically remote location. By continuously transferring log data from the primary database to the backup and applying the changes there, the backup can be kept (almost) up-to-date.

Cross-references

- ▶ ACID Properties
- ▶ Application Recovery
- ▶ Backup and Restore
- ▶ Buffer Management
- ▶ Crash Recovery
- ▶ Database Repair
- ▶ Multi-Level Recovery and the ARIES Algorithm
- ▶ RAID

Recommended Reading

1. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1983.
2. Haerder T. and Reuter A. Principles of transaction-oriented database recovery. ACM Comput. Surv., 15(4):287–317, 1983.

Logging/Recovery Subsystem

Andreas Reuter[1,2]
[1]EML Research gGmbH Villa Bosch, Heidelberg, Germany
[2]Technical University Kaiserslautern, Kaiserslantern, Germany

Synonyms

Audit trail; Log component; Log manager; Recovery manager

Definition

The logging/recovery subsystem (LRS) of a DBMS is responsible for implementing the fault tolerance mechanisms needed to support database transactions.

The log component stores the information needed to undo the updates performed by a transaction in case it has to be rolled back, either to an internal save point or to the beginning. It also stores information needed to re-apply the updates of committed transactions to the database in case they are (partially) lost due to a system crash or after a storage media failure. In addition, the log component keeps track of all relevant state transitions such as begin-transaction, prepare, commit, abort, checkpoint, etc. The log is the first resource that is activated when restarting the database after a crash [1]. The recovery component orchestrates the activities needed to repair the database, depending on the situation. For example, after a crash, it first locates the last log record written and then reads the log backward, initiating undo for the operations of all transactions for which no "prepare" or "commit" entry has been found. Going backward, it looks for two specific records: The "begin" record of the oldest incomplete transaction – this is where undo stops, and the youngest checkpoint record. From that record, the recovery will read the log in forward direction and initiate a redo for the operations of all transactions that were completed before the crash. The log also plays a crucial role in implementing the two-phase commit protocol, and it can be used to detect security breaches.

Key Points

The log is the starting point of any recovery activity, so it has to be very reliable; therefore, the log devices are stored on devices that will retain their data in case of a power failure – typically disks, with current technology. Since all update operations, some read operations and a number of other activities create records to the log, it has to be very fast in order not to create a bottleneck. For that reason, the log is divided in to two portions: an online log containing all information that would be needed to support crash recovery, and an archive log that contains all information needed to recover from the loss of storage media. Data no longer needed in the online log is continuously moved to the archive by a background process that does not affect the overall system performance.

The recovery manager uses the log component for implementing some variation of the "write ahead log" protocol [2]. It basically states that before any update is applied to the database that may need to be rolled back, the data supporting the undo must be written to the log. Before a transaction is committed, all data needed to repeat the updates must be written to the log. Neither the log manager nor the recovery manager "understand" the structure of the log records. They are created and used by the resource managers that implement the objects (e.g., tuples, B-trees, queues). The log manager writes log records on behalf of those resource managers, and the recovery managers feeds them back to the resource managers when they are needed for undo and/or redo.

Cross-references

- ▶ Buffer Manager
- ▶ I/O-Subsystem
- ▶ Storage Resource Management
- ▶ Transaction Manager

Recommended Reading

1. Gray J. and Reuter A. Transaction Processing – Concepts and Techniques. Morgan Kaufmann, San Mateo, CA, 1993.
2. Härder T. and Reuter A. Principles of transaction oriented database recovery - a taxonomy. ACM Comput. Surv., 15 (4):287–317, 1983.

Logic of Time

- ▶ Time in Philosophical Logic

Logical Database Design: from Conceptual to Logical Schema

Alexander Borgida[1], Marco A. Casanova[2], Alberto H. F. Laender[3]
[1]Rutgers University, Piscataway, NJ, USA
[2]Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
[3]Federal University of Minas Gerais, Belo Horizonte, Brazil

Synonyms

Logical schema design; Data model mapping

Definition

Logical database design is the process of transforming (or mapping) a conceptual schema of the application domain into a schema for the data model underlying a particular DBMS, such as the relational

or object-oriented data model. This mapping can be understood as the result of trying to achieve two distinct sets of goals: (i) representation goal: preserving the ability to capture and distinguish all valid states of the conceptual schema; (ii) data management goals: addressing issues related to the ease and cost of querying the logical schema, as well as costs of storage and constraint maintenance. This entry focuses mostly on the mapping of (Extended) Entity-Relationship (EER) diagrams to relational databases.

Historical Background

In the beginning, database schema design was driven by analysis of the prior paper or file systems in place in the enterprise. The use of a conceptual schema, in particular Entity Relationship diagrams, as a preliminary step to logical database design was proposed by Chen in 1975 [2,3], and the seminal paper on the mapping of EER diagrams to relational databases was presented by Teorey et al. in 1986 [7]. Major milestones in the deeper understanding of this mapping include [1,6], which separate the steps of the process, and pay particular attention to issues such as naming and proofs of correctness. Among others, the correct mapping of subclass hierarchies requires a careful definition of table keys [5], and the *maintenance* of optimized relational representations of ER schemas is discussed in [4].

Foundations

The mapping from an Extended Entity Relationship schema to a relational (logical) schema handles the issue of representing the different states of the conceptual schema through a function that provides for every "object set" O (entity set or relationship set) in the EER schema a relational table `T_O`. The data management-related issues are handled by merging, partitioning or otherwise reorganizing the tables obtained in the previous step, while making sure that there is no loss of information.

To begin with, some notation and assumptions concerning the conceptual schema expressed in EER notation are required. A unique set of identifying attributes $id(E)$ is assumed to be available for every strong entity set E (i.e., one that is not a weak entity or a subclass). Each object set O participating in a relationship set R can be marked as being: *total*, indicating that each instance of O must participate in at least one relationship instance of R; *functional*, indicating that an instance of O can participate in at most one relationship instance of R; and as playing a particular *role* in the relationship, if O (or its sub/super-class) can participate in other relationships or as different arguments of R. For every relationship set R, $id(R)$ is assumed to return a subset of participants that uniquely determine each relationship instance. This is needed in examples such as "Exactly one faculty advises each student for each major area," where id`(advises)`$=$ `{Student,Major}`, because students can major in several areas. Of course, if the relationship set R has one functional participant O, then $id(R)$ returns O; when there are multiple functional participants, $id(R)$ is assumed to return one that is total, if available. Entities can be organized in a ISA/subclass hierarchy, with the ability to declare subclasses as *disjoint*, and/or *covering* the superclass.

In a relational schema, for each table T, one must specify its attributes/columns, its primary key, any foreign key referential constraints, and non-null constraints on columns. The notation $T[\underline{K}Y]$ is used to refer to a table named T, with columns KY and primary key K; $key(T)$ returns K.

The Basic E2R Mapping

The initial mapping from entity and relationship sets in the conceptual schema to relational tables (referred to as the E2R mapping) is defined recursively as follows: For a strong entity set E, the attributes of E form the columns of table `T_E`, and its key is set to $id(E)$. For example, for entity set `Student`, with attributes `sid`, `name` and `age`, and id`(Student)=sid`, the E2R mapping will generate a table `T_Student` `[`$\underline{\texttt{sid}}$`,name,age]`.

To preserve first-normal form `T_E` does not include multi-valued attributes of E. For any such attribute M of E, one adds a separate table `T_EM`, whose attributes are those in $id(E)$ plus a new attribute $\hat{M}$ that holds the individual values that occur in M; the full set of columns forms the key of this table, and a foreign key constraint is added from column $id(E)$ of `T_EM` to `T_E`. Thus, if the entity set `Student` has a multi-valued attribute `phone`, then the E2R mapping will generate a table `T_StudentPhone[`$\underline{\texttt{sid,phone}}$`]`, with `sid` being a foreign key with respect to `T_Student`.

If E is a subclass of some class F, then the columns of `T_E` consist of the attributes of E together with those in key`(T_F)` – the key "inherited" from `T_F`.

Therefore *key*(T_F) is the key of T_E, and a foreign key reference to T_F is needed to ensure that every subclass instance is in the super-class. For example, supposing that GradStudent is given as a subclass of Student, then T_GradStudent will have, among others, a column sid, which will also be its key and a foreign key referencing T_Student. In case subclasses are disjoint or cover the super-class, appropriate SQL assertions need to be added to check these constraints.

For a relationship set *R*, T_R has as attributes all the attributes of *R*, as well as the union of the sets of attributes X_O=*key*(T_O) of all object sets *O* participating in *R*; for each such set of attributes X_O, a foreign key constraint from T_R to T_O is added, in order to avoid dangling references. Moreover, a NOT NULL constraint is added to the corresponding columns of X_O. The keys of the tables generated from the object sets in *id*(*R*) jointly become the key of T_R. For example, suppose that admits is defined as a relationship set with participating entity sets Professor and Student, plus attribute Year. Assuming that *id*(Professor)=pid, then the relational representation of admits would be T_admits[pid,sid,year], with pid and sid being foreign keys that reference T_Professor and T_Student, respectively. Additional constraints found in some EER schemas, such as numeric lower and upper bounds on relationship participation (e.g., a child has between 2 and 2 parents), can only be enforced using general SQL assertions.

For a weak entity set *W*, T_W includes the attributes of *W*, any attributes of the identifying relationship set of *W*, and the identifying attributes *key*(T_E) of the entity set *E* that "owns" *W*. The key of T_W is the union of the local identifier *id*(W) of *W* with the key attributes *key*(T_E) of *E*. A foreign key constraint from T_W to T_E must also be added. For example, suppose that Section is specified as a weak entity set with respect to Course, with local identifier sectionNr and identifying relationship set sectionOf; furthermore, assume that *key*(T_Course)=courseNr. Then, the relational representation of Section would be T_Section[courseNr,sectionNr], with courseNr as a foreign key referencing T_Course.

Refinements

In the creation of tables for relationship sets and for weak entity sets, special attention needs to be paid when duplicate column names arise (because the same object can be involved in a relation in multiple ways). In this case, role names need to be used to disambiguate the columns. For example, if *id*(Professor)=*id*(Student)=ssn (because both are subclasses of Person say), then T_admits should have columns named prof_ssn and student_ssn, or admitter_ssn and admitted_ssn.

The treatment of subclass hierarchies which are not trees and where subclasses may inherit different identifiers from different parents also requires special care, and is discussed in [5].

The previous constructions provide a relational schema that allows every instance of the conceptual schema to be captured precisely. Relational schema restructuring by relation merging and sometimes partitioning is then undertaken for a number of reasons. The cardinal rule of all techniques is the need to be able to recover precisely the original relation instances from the merged instance (viz. lossless join). An additional rule observed in the techniques here is to avoid duplicating information in ways that lead to "update anomalies."

Table Merging

The most familiar technique replaces tables by their outer join, with the goal of making it easier to express and evaluate queries by avoiding the join. For example, one can merge table student[sid,name,age] with minorsIn[sid,minor], to obtain student2 [sid,name,age,minor]. Conceptually, such a change usually merges a functional relationship with the entity it is about, or a sub-class with its super-class. The main disadvantage of this change is the need to store null values as part of the outer join (in the above example, for students who do not have a minor).

The rule being applied in this case can be stated as:

Rule 1: Tables $T[\underline{K}X]$ and $R[\underline{\hat{K}}Y]$, where $\hat{K}$ is a foreign key referencing *T*, can be replaced by table $T_R[\underline{K}XY]$, whose intended use is as the left outer join of *T* and *R*.

Applying the above transformation, one must be mindful of a number of potential problems. First, identical column names occurring in *X* and *Y* cause conflicts, which must be avoided by renaming. Second, this merge may prevent making previously possible distinctions when there is no guarantee that for every tuple in table *T* there is a corresponding tuple in *R* referencing it. For example, in the original schema one can distinguish the case of a student who has a minor that is not known (represented by a tuple with NULL in

column minor of table minorsIn) from the case of a student without a minor (represented by the absence of a the student's sid in minorsIn); but in student2 both cases have NULL in the minor column. To capture such distinctions one can add a Boolean attribute isInMinorsIn? to table student2. This technique becomes essential when merging the table of a subclass (e.g., T_GradStudent) into that of the superclass (e.g., T_Student) in case all the additional attributes of the subclass (e.g., advisor) may have null values, and therefore cannot be used to detect membership in the subclass.

Third, previously easy-to-state constraints may now become more convoluted. For example, if graduate students must have both an advisor and a department attribute (hence these columns have NOT NULL constraints in T_GradStudent), then since nulls must be allowed into these columns of T_Student after the merge, one must ensure (using an SQL check constraint) that nulls in the two columns *correlate*. Moreover, foreign key references to table T_GradStudent now become references to T_Student, and must be supplemented by SQL assertions verifying that some attribute associated with graduate students has a non-null value.

Table Partitioning

Tables can also be reorganized by so called "horizontal splitting" as stated by the following rule:

Rule 2: Table $T[\underline{K}X]$ can be replaced by tables $T_1[\underline{K}X]$, $T_2[\underline{K}X]$,...,with the intended use of T_1, T_2,... being as a partition of the tuples in T.

In logical (as opposed to physical) schema design, the partition tables usually have semantic interpretation as subclasses of relationship sets (e.g., T_admitted replaced by T_currentlyAdmitted and T_previouslyAdmitted) or of entitie sets (e.g., T_Student[sid,name,age] replaced by gradStudent$_0$[sid,name,age], undergradStudent$_0$[sid,name,age], and student$_0$[sid,name,age] – the latter with students that are neither undergraduate nor graduate). Note that after such a split, one can merge gradStudent$_0$[sid,name,age] with gradStudent[sid,office] to get another variant of mapping subclass hierarchies to tables; this one is particularly good for cases when the subclasses cover the superclass.

Table partitioning can be seen as encoding into each table selection criteria, which therefore once again facilitates query statement and evaluation. The down side is that once again built-in constraints, such as foreign keys, now need to be stated as more complex SQL inter-table assertions.

Mapping from Non-ER Conceptual Schemas

UML class diagrams are increasingly popular for the specification of conceptual schemas. The correspondences "class" ≓ "entity set," "association" ≓ "relationship set" make it easy to reformulate the E2R mapping as a UML-to-Relational (U2R) mapping. Higher arity relationships (such as assignedTo(Professor,Course,Semester)) need to be reified in UML (i.e., represented as classes of objects Assignment related by functional associations f_1, f_2 and f_3 to Professor, Course and Semester respectively), but these end up producing a similar relational schema as E2R because tables T_f_i are merged into T_Assignment; the artificial key *id*(Assignment) should however be removed.

As in the above example, the main difficulty in U2R is dealing with identifying ("key") attributes for entities, which are not mandated by the UML data model, since it assumes objects have intrinsically unique identity.

Key Applications

Database Design

The mapping of the conceptual schema into a logical schema is the central step of the database design process. A carefully crafted mapping will guarantee that the logical schema correctly represents the application domain that the conceptual schema models.

Future Directions

The publication of the SQL:2003 language standard provides additional mapping opportunities, exploiting some of the object-oriented features of the language.

As a simple example, SQL:2003 introduces the MULTISET data type, which can be used to avoid defining a separate table to accommodate multi-valued attributes of entity or relationship sets.

More importantly, SQL:2003 supports the declaration of an *"identity attribute"* for a table (designated with the special keyword IDENTITY). The value of an identity attribute is unique and automatically generated whenever a new row is inserted into the table. This construct is useful to the E2R and, especially, to the

U2R mapping process since it avoids the creation of an *artificial* key for entity *E* by adding an identity attribute for table T_E.

URL to Code

DBDesigner (http://fabforce.net/dbdesigner4/) is an open-source database design system, available from fabFORCE.net, which integrates EER modeling with the derivation and maintenance of a relational schema for mySQL.

Cross-references

- ▶ Extended Entity-Relationship Model
- ▶ Functional Dependency
- ▶ Information Capacity

Recommended Reading

1. Casanova M.A., Tucherman L., and Laender A.H.F. On the design and maintenance of optimized relational representations of entity-relationship schemas. Data Knowl. Eng., 11(1):1–20, 1993.
2. Chen P.P. The entity-relationship model: toward a unified view of data. In Proc. 1st Int. Conf. on Very Large Data Bases, 1975.
3. Chen P.P. The entity-relationship model – toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
4. da Silva A.S., Laender A.H.F., and Casanova M.A. An approach to maintaining optimized relational representations of entity-relationship schemas. In Proc. 15th Int. Conf. on Conceptual Modeling, 1996, pp. 292–308.
5. da Silva A.S., Laender A.H.F., and Casanova M.A. On the relational representation of complex specialization structures. Inf. Syst., 25(6–7):399–415, 2000.
6. Markowitz V.M. and Shoshani A. Representing extended entity-relationship structures in relational databases: a modular approach. ACM Trans. Database Syst., 17(3):423–464, 1992.
7. Teorey T.J., Yang D., and Fry J.P. A logical design methodology for relational databases using the extended entity-relationship model. ACM Comput. Surv., 18(2):197–222, 1986.

Logical Foundations of Web Data Extraction

Christoph Koch
Cornell University, Ithaca, NY, USA

Definition

Several wrapper programming languages for extracting information from Web pages have been based on logic, specifically on fragments of datalog. This entry shows how logical languages can be used for Web information extraction, and surveys expressiveness and complexity aspects of a foundational logical wrapping language, monadic datalog.

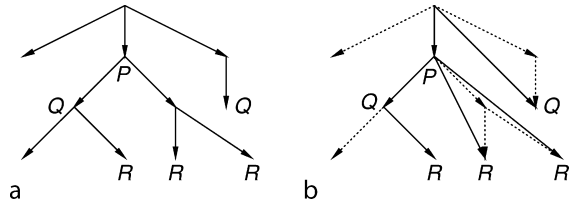
Historical Background

A substantial amount of research has studied the problem of learning wrapper programs from examples (see Wrapper Induction). Unfortunately, it is now known that the expressive power of learnable wrappers is fundamentally limited. This has motivated further work on visual wrapper programming languages, which simplify and speed up wrapper definition. Visual wrapping is now supported by several implemented systems (cf. XWrap [3] and W4F [4]; Lixto [1], a commercial product). The Lixto project was the first to emphasize and study expressiveness of visual wrapper languages. Its approach is based on a datalog-like language, ELog, whose core was shown to be expressively equivalent to monadic second-order logic over finite node-labeled trees, a natural yardstick for such languages. The datalog approach to wrapping has since then be adopted by other researchers; for instance, the recent XLog wrapper language [5] is closely related to ELog.

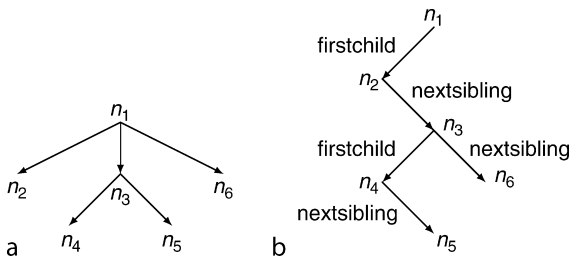
Foundations

Information Extraction Functions and Wrappers

The core notion on which logic-based wrapping is based is that of an *information extraction function*, which takes a labeled unranked tree (representing a Web document) and returns a subset of its nodes. In other words, an information extraction function is a unary query. A wrapper is a program which implements one or several such functions, and thereby assigns unary predicates to document tree nodes. Based on these predicate assignments and the structure of the input document viewed as a tree, a new tree can be computed as the result of the information extraction process in a natural way, along the paths of the input tree but using the new labels and omitting nodes that have not been assigned a new label. (see Fig. 1 for an example of such a transformation.) That way, the nodes of a document tree can be re-labeled, and some can be dropped as irrelevant, but it is not possible to significantly transform the original tree structure. This coincides with the intuition that a wrapper may change the presentation of relevant information, its packaging or data model (which does not apply in the case of *Web wrapping*), but should not handle substantial data



Logical Foundations of Web Data Extraction. Figure 1. Tree annotated with predicates *P*, *Q*, and *R* defined by information extraction functions (a), and wrapping result (b). (Original node labels of the input tree are not shown.)



Logical Foundations of Web Data Extraction. Figure 2. (a) An unranked tree and (b) its representation using the binary relations "firstchild" (↙) and "nextsibling" (↘).

transformation tasks. There is wide agreement that wrapping should exclude operations such as joins which are not necessary for information extraction but may be part of data transformation and integration tasks that may follow extraction.

A main goal is thus to find a language for specifying expressive unary queries over trees which nevertheless can be efficiently computed.

Tree Structures

Unranked finite ordered trees with node labels from a finite set of symbols Σ correspond closely to parsed HTML or XML documents. In an unranked tree, each node may have an arbitrary number of children. An unranked ordered tree can be considered as a structure of relational schema

$$\tau_{ur} = \langle \text{dom},\ \text{root},\ \text{leaf},\ (\text{label}_a)_{a\in\Sigma},\ \text{firstchild},\ \text{nextsibling},\ \text{lastsibling}\rangle$$

where "dom" is the set of nodes in the tree, "root", "leaf", "lastsibling", and the "label$_a$" relations are unary, and "firstchild" and "nextsibling" are binary. All relations are defined according to their intuitive meanings. "root" contains exactly one node, the root node. "leaf" consists of the set of all leaves. "firstchild(n_1, n_2)" is true if n_2 is the leftmost child of n_1; "nextsibling(n_1, n_2)" is true if, for some i, n_1 and n_2 are the ith and $(i + 1)$th children of a common parent node, respectively, counting from the left (see also Fig. 2). label$_a$(n) is true if n is labeled a in the tree. Finally, "lastsibling" contains the set of rightmost children of nodes.

Monadic Datalog

Monadic datalog is obtained from full datalog (see Datalog) by requiring all intensional predicates to be unary. A unary query is a function that assigns a predicate to some elements of dom (or, in other words, selects a subset of dom). For monadic datalog, one obtains a unary query by distinguishing one intensional predicate as the *query predicate*. In the remainder of this entry, a monadic datalog query will always be understood to be a unary query specified as a monadic datalog program with a distinguished query predicate. Of course, monadic datalog allows for the the definition of multiple unary queries within a single program; thus a set of information extraction functions that defines a wrapper can be given through a single monadic datalog program.

Monadic second-order logic (MSO) is obtained from second-order logic by requiring all second-order quantifiers to range over sets (i.e., unary relations). A unary MSO *query* is defined by an MSO formula φ with one free first-order variable. Given a tree t, it evaluates to the set of nodes $\{x \in \text{dom} \mid t \vDash \varphi(x)\}$.

It has been argued in [2] that unary queries in monadic second-order logic (MSO) over trees are an appropriate expressiveness yardstick for information extraction functions. MSO over trees is well-understood theory-wise and quite expressive. It has also been used as an expressiveness yardstick for node-selecting XML query languages.

By restricting the structures considered to trees, monadic datalog acquires a number of nice properties. First, the query evaluation complexity is linear in the size of the data and of the program:

Theorem 1 ([2]). *Over τ_{ur}, monadic datalog has $O(|\mathcal{P}|*|dom|)$ combined complexity (where $|\mathcal{P}|$ is the size of the program and $|dom|$ the size of the tree).*

This is is stark contrast to full datalog, which is EXPTIME-complete w.r.t. combined complexity over

arbitrary finite structures and NP-complete over trees. Monadic datalog has some other nice properties which have been studied; in particular, the containment and boundedness properties – which are both considered relevant to query optimization – for monadic datalog are decidable, while they are undecidable for full datalog.

A unary query over trees is MSO-definable exactly if it is definable in monadic datalog.

Theorem 2 ([2]). *A unary query over τ_{ur} unranked ordered finite trees is MSO-definable if and only if it is definable in monadic datalog.*

Theorem 2 asserts that monadic datalog programs can define "universal" properties over trees, such as that a certain fact holds everywhere or nowhere in a tree. This may seem somewhat unintuitive because monadic datalog does not feature negation. Still, it follows from the fact that the relations defining the tree in τ_{ur} allow us to traverse the tree (starting from its "ends" such as the leaves or the root) using a recursive program and to compute universal properties along the way.

Example 1. Consider the problem of selecting all those nodes from an HTML tree which do *not* contain an HTML "table" in their subtrees. A monadic datalog program for this (with query predicate Q) can be defined as follows.

$$
\begin{aligned}
\mathit{NoTableBelow}(x) &\leftarrow \mathit{leaf}(x).\\
\mathit{NoTableBelow}(x) &\leftarrow \mathit{firstchild}(x,y), \mathit{NoTable}(y).\\
\mathit{NoTableRight}(x) &\leftarrow \mathit{lastsibling}(x).\\
\mathit{NoTableRight}(x) &\leftarrow \mathit{nextsibling}(x,y), \mathit{NoTable}(y).\\
\mathit{NoTable}(x) &\leftarrow \overline{\mathit{label}_{\mathit{table}}}(x), \mathit{NoTableBelow}(x),\\
&\qquad \mathit{NoTableRight}(x).\\
Q(x) &\leftarrow \overline{\mathit{label}_{\mathit{table}}}(x), \mathit{NoTableBelow}(x).
\end{aligned}
$$

Here it may either be assumed that there is a predicate $\overline{\mathrm{label}_{\mathit{table}}}$ true for those nodes not labeled "table" (this predicate then needs to be added to τ_{ur}) or $\overline{\mathrm{label}_{\mathit{table}}}$ can be defined in monadic datalog by the rules $\{\overline{\mathrm{label}_{\mathit{table}}}(x) \leftarrow \mathrm{label}_l(x). \mid l \in \Sigma, l \neq \text{"table"}\}$.

Note that both $\overline{\mathrm{label}_{\mathit{table}}}$ and *NoTableBelow* are true for a node if it does not contain a "table" in its subtree, while *NoTable* is true for a node if the same holds (i.e., it does not contain a "table" in its subtree) in the binary-tree model of Fig. 2b.

Each monadic datalog program over trees can be efficiently rewritten into an equivalent program using only very restricted syntax. This motivates a normal form for monadic datalog over trees.

Definition 1. A monadic datalog program $\mathcal{P}$ over τ_{ur} is in *Tree-Marking Normal Form* (TMNF) if each rule of $\mathcal{P}$ is of one of the following four forms:

$$
\begin{aligned}
&(1)\ p(x) \leftarrow p_0(x).\\
&(2)\ p(x) \leftarrow p_0(x_0), R(x_0,x).\\
&(3)\ p(x) \leftarrow p_0(x_0),\ R(x,x_0).\\
&(4)\ p(x) \leftarrow p_0(x),\ p_1(x).
\end{aligned}
$$

where the unary predicates p_0 and p_1 are either intensional or from τ_{ur} and R is a binary predicate from τ_{ur}.

In the next result, the schema for unranked trees may extend τ_{ur} to include the natural child relation – likely to be the most common form of navigation in trees.

Theorem 3 ([2]). *For each monadic datalog program $\mathcal{P}$ over $\tau_{ur} \cup \{child\}$, there is an equivalent TMNF program over τ_{ur} which can be computed in time $O(|\mathcal{P}|)$.*

Syntax as simple as that of TMNF is important for visual wrapping. A single rule in monadic datalog may still consist of an arbitrary number of joins involving the binary relations of τ_{ur}. TMNF is much simpler and a visual rule definition process is not hard to define (cf. [1]). Nevertheless, TMNF has the full power of MSO and admits efficient evaluation. TMNF is the core of the ELog language of the Lixto system [1].

Key Application

Web Information Extraction; XML Query Languages.

Cross-references

- ▶ Datalog
- ▶ Wrapper
- ▶ Wrapper Generator
- ▶ Wrapper Induction

Recommended Reading

1. Baumgartner R., Flesca S., and Gottlob G. Visual web information extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
2. Gottlob G. and Koch C. Monadic datalog and the expressive power of web information extraction languages. J. ACM, 51(1):74–113, 2004.

3. Liu L., Pu C., and Han W. XWRAP: An XML-enabled wrapper construction system for web information sources. In Proc. 16th IEEE Int. Conf. on Data Engineering, 2000, pp. 611–621.
4. Sahuguet A. and Azavant F. Building intelligent web applications using lightweight wrappers. Data Knowl. Eng., 36(3):283–316, 2001.
5. Shen W., Doan A., Naughton J.F., and Ramakrishnan R. Declarative information extraction using datalog with embedded extraction predicates. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1033–1044.

Logical Models of Information Retrieval

Fabio Crestani
University of Lugano, Lugano, Switzerland

Definition

Logical models of Information Retrieval (IR) are defined as those that follow a *logical definition of relevance*. For Cooper logical relevance is defined as "logical consequence." To make this possible both queries and documents need to be represented by sets of declarative sentences. The query is represented by two formal statements called "component statements" of the form p and $\neg p$. A subset of the set of stored sentences is called "premiss set" if and only if the component statement is a logical consequence of that subset. A "minimal premiss set" for a component statement is one that is as small as possible. Logical relevance is therefore defined as a two-place relation between stored sentences and the query represented as component statements:

- "A stored sentence is logically relevant to (a representation of) an information need if and only if it is a member of some minimal premiss set of stored sentences for some component statement of that need."

This definition of relevance is essentially just a proof-theoretic notion that has been later generalized to be applicable to information needs involving more than one component statement.

An early common belief was that the logical implication needed to capture relevance was not the classical material implication. The reasons why the use of the classical material implication is not appropriate for IR is in the definition of material implication itself (see the historical background section). The idea that a non-classical form of logical implication was needed for defining relevance was proposed by Van Rijsbergen in the form of the *logical uncertainty principle*, which is defined as follows:

- "Given any two sentences x and y; a measure of the uncertainty of $y \rightarrow x$ related to a given data set is determined by the minimal extent to which we have to add information to the data set, to establish the truth of $y \rightarrow x$."

This principle made an explicit connection between non-classical logics and IR uncertainty modelling. However, when proposing the above principle, Van Rijsbergen was not specific about which logic and which uncertainty theory to use. As a consequence, various logics and uncertainty theories have been proposed and investigated. The choice of the appropriate logic and uncertainty mechanisms has been central to logical IR modeling, leading to a number of different approaches (see the Foundations section).

Historical Background

Relevance is one of the most important, if not "the fundamental," concept in the theory of IR. The concept arises from the consideration that if a user of an IR system has an information need, then some information stored in some documents in a document collection may be "relevant" to this need.

A logical definition of relevance was considered for the first time in the context of IR by Cooper in 1971 [4]. For Cooper, *logical relevance* was another name for topic-appropriateness, and he addressed the problem of giving a definition of logical relevance for IR by analogy with the same problem in question-answering systems. The analogy goes only as far as having questions with a yes-no (true-false) type of answer, and while Cooper's work started by analyzing question-answering systems, later he abandoned the analogy. Relevance is defined by Cooper as "logical consequence." To make this possible both queries and documents need to be represented by sets of declarative sentences. In the case of a yes-no query, the query is represented by two formal statements of the form p and $\neg p$. The two statements representing the query are called "component statements." A subset of the set of stored sentences is called "premiss set" if and only if the component statement is a logical consequence of that subset. A "minimal premiss set" for a component statement is one that is as small as possible in the sense that if any of its members were deleted, the component statement would no longer be a logical consequence of the set. Logical relevance is defined as a two-place

relation between stored sentences and the query represented as component statements (see definitions section).

This definition of relevance is essentially just a proof-theoretic notion that has been generalised to be applicable to information needs involving more than one component statement. Although logical relevance was initially defined only for sentences, it can be easily extended to apply to stored documents: a document is relevant to an information need if and only if it contains at least one sentence which is relevant to that need.

Cooper also attempted to tackle a generalization of such a definition to natural language queries and documents. However, without a formalized language, no precise definition of the logical consequence relation is at hand, and thus one loses a precise definition of relevance. The problems of ambiguity and vagueness of natural language deny the possibility of extending the previous logical notion of relevance, despite the fact that the general idea of implication in natural language is a reasonably clear one. The definition of relevance, so far as natural language in concerned, is only a definition-in-principle – a conceptual definition – but not yet defined on a mathematical level.

Finally, Cooper also tried to tackle the problem of having "degrees of relevance," or as he wrote: "shades of grey instead of black and white." The idea was to extend the system of deductive reasoning used to access logical relevance to a system of plausible reasoning. Cooper argued that plausible or probabilistic inference was not as well defined as deductive inference, even for formalized languages. However, he added that when such tools are formalized enough then this development would become a "sensible and indeed inescapable idea," because it would enable the ranking of documents according to an estimated probability of relevance. What he proposed was to assign a higher probability of relevance to a sentence or a document that has greater probability of belonging to a residual minimal premiss set.

Cooper was the first to associate the topic-appropriateness sense of relevance with logical implication and recognized the importance of evaluating the uncertainty of such implication to rank documents in relation to their estimated measure of relevance. Many other researchers followed this idea proposing the use of different logics to capture relevance. In fact, the use of logic to build IR models enables one to obtain models that are more general than earlier well known IR models. Indeed, some logical models are able to represent within a uniform framework various features of IR systems, such as hypermedia links, multimedia content, users knowledge, cross-lingual, and structured documents. It also provides a common approach to the integration of IR systems with logical database systems. Finally, logic makes it possible to reason about an IR model and its properties. This latter possibility is becoming increasingly important since conventional evaluation methods, although good indicators of the effectiveness of IR systems, often give results which cannot be predicted, or satisfactorily explained.

An early common belief was that the logical implication needed to capture relevance was not the classical material implication. The reasons why the use of the classical material implication $d \supset q$ is not appropriate for IR is in the definition of material implication itself, and there are many ways of explaining why material implication is not suitable for IR. For reasons of brevity, these arguments will not be repeated here. It suffices to say that the material implication is acceptable only for the Boolean model of IR , where it expresses the "modus ponens." In the Boolean model if one observes d (that is if d is true) and if $d \rightarrow q$ is true (that is if all the query terms are all in the documents) then q is also true and thus one retrieves d. Only in this case if $d \rightarrow q$ is equivalent to $d \supset q$. In all other cases this would not hold ant it would necessary to find a way to define a suitable notion of implication that measures the extent in which the implication holds.

More recently, the thought that logic by itself could not fully model IR started to find a number of followers. In fact, in determining the relevance of a document to a query, the success or failure of an implication relating the two is not enough. It is necessary to take into account the *uncertainty* inherent in such an implication. The introduction of uncertainty can also be motivated from the consideration that a collection of documents cannot be regarded as a consistent and a complete set of statements. In fact, documents in the collection could and often do contradict each other in any particular logic, and not all the necessary knowledge is available. To cope with uncertainty a logic for *uncertain inference* was introduced. In fact, if $d \rightarrow q$ is uncertain, then one can measure its degree of uncertainty by $P(d \rightarrow q)$.

In 1986, Van Rijsbergen proposed the use of a non-classical conditional logic for IR [16]. This would enable the evaluation of $P(d \rightarrow q)$ using the *logical uncertainty principle* (see definitions). This principle

was the first attempt to make an explicit connection between non-classical logics and IR uncertainty modelling. However, when proposing the above principle, Van Rijsbergen was not specific about which logic and which uncertainty theory to use. As a consequence, various logics and uncertainty theories have been proposed and investigated. This lead to a number of different approaches being proposed over the years.

Foundations

In the following, a brief overview of the most important logical models of IR is provided. It divides them into three classes: local models, local-uncertainty models, and meta-models.

Logical Models

The best known class of logical models of IR is that of the Boolean models, but their inability to capture the uncertainty inherent in the IR process (except by some extension of these models) has always confined them to the margins of modern IR.

Some logical models are able to capture the uncertainty of IR process, mainly in two ways: qualitatively by the logic itself (for example, via default rules, nonmonotonicity, or background conditions), or quantitatively by adding an uncertainty theory to the logic (for example, fuzzy logic). A first example of this class of models is represented by models based on Modal Logic and Conceptual Graphs. Modal Logic adopts the notion of possible worlds that correspond to the interpretations in classical logic, but which are connected to each other via an accessibility relation. The evaluation of the truth of a proposition is with respect to a possible world, and may involve the evaluation of the truth of the proposition in connected worlds. *Modal Logic* was first used to develop a logical model for IR by Nie [12]. Documents are worlds, and queries are formulae. A document represented by a world d is relevant to a query represented by a formula q if q is "true" in d, or if it is true in a world d' accessible from d. The accessibility relation captures the transformation of documents; the fact that the world d is connected to the world d' is interpreted as d being transformed into d'. The accessibility relationship can have different properties. For example, transitivity, meaning that if a world d is related to a world d', which is itself related to a world d'', then the world d is also related to the world d''. Consider the example of a hypertext system. Using Modal Logic, worlds can represent texts (nodes) and the accessibility relation can represent the links between the texts. The model also allows the transformation of both the query and the data set. One query can be transformed into another one using, for example, thesaural information. Query transformation is not a new approach in IR (e.g., query expansion). The novelty is that the transformation process can be formally represented, and hence reasoned upon. Transforming a data set can capture the modeling of a user's state in the retrieval process. The data set can be transformed until it reaches one that reflects the user's state.

Another set of logical models belonging are based on Situation Theory and the related Channel Theory. Situation Theory is a theory of information that provides an analysis of the concept of information and the manner in which cognitive agents handle and respond to the information picked up from their environment. The theory defines the nature of information flow and the mechanisms that give rise to such a flow. A document is a situation s and the query is a type ϕ. The document is relevant to the query if there exists a flow of information from a situation s to a situation s' such that $s' \vDash \phi$. The nature of the flow depends on the so-called "constraints" which capture semantic relationships. Flows of information do not always materialize because of the unpredictable nature of situations, thus flows are often uncertain. In Situation Theory, an uncertain flow is modeled by a conditional constraint of the form $\phi \rightarrow \psi | B$, which highlights the fact that $\phi \rightarrow \psi$ holds if some background conditions captured within B are met. If the background conditions are satisfied, the corresponding flow arises. However, it is often the case that two situations are systematically related to each other, by way of a flow of information. Therefore, in addition to constraints, there are relationships that link situations. The concept of a channel is introduced to express the relationships, by way of an information flow, between two situations. *Channel theory* defines formally channels, together with the mathematical properties that support the flow of information. The use of Situation and Channel theory to model IR has been investigated in [17], where the connection between IR, logic, probability and information containment was made. It was indicated that the use of channels present many potentials for theoretical IR modelling because they can apply to various IR processes present in advanced IR systems.

A successful class of logical models is based on *Terminological logics*. This family of logics come from the area of artificial intelligence, in particular, knowledge representation. Terminological logics derive from a large group of knowledge representation language (such as, for example, KL-ONE) based on semantic networks and inspired by the notion of frames. They provide object-oriented flavored representations. The use of terminological logic for IR was proposed in [11]. There, documents are represented by individual constants, whereas a class of documents is represented as a concept. Queries are described as concepts. Given a query represented by a concept Q, the retrieval task is to find all those documents d such that $Q[d]$ holds. The evaluation of $Q[d]$ uses the set of assertions describing documents, that is, one is not evaluating whether $d \rightarrow q$, but rather whether individual d is an instance of the class concept Q.

An important class of logical models is based on Belief Revision. *Belief Revision* provides a way to formalize changes done to a knowledge base after the arrival of new information. The use of belief revision in IR was attempted in [10] as a way to compute the similarity of a document to a query for retrieval purpose. The Dalal's revision operator was chosen for implementing the belief revision process, since it provides an order among proposition interpretations, where propositions model index terms. The ordering is used to formulate a similarity measure between a document and a query (expressed as formulae) based on the number of revisions necessary for the document to "reach" the query. It should be noted that both documents and queries can have, each, several interpretations, so normalization becomes necessary.

Finally, there is the class of models based on *Fuzzy logic*. This is a formal framework that is well suited to model vagueness and imprecision. In IR it has been successfully employed at several levels in particular for the definition of a superstructure of the Boolean model, with the appealing consequence that existing Boolean IR systems can be improved without redesigning them completely [2]. Through these extensions the gradual nature of relevance of documents to user queries can be modeled.

Logical-Uncertainty Models

Logical-uncertainty models are based on an uncertainty theory (for instance, probability theory, semantic theory, imaging) that is defined on a logical basis. They enable a more complex definition of relevance than other IR models (than probabilistic relevance model, for instance, which are based mainly upon statistical estimations of the probability of relevance). With logical-uncertainty models, information not present in the query formulation may be included in the evaluation of the relevance of a document. Such information might be domain knowledge, knowledge about the user, user's relevance feedback, or other.

The main example of logical-uncertainty models of IR is represented by models based on Probability Theory. In IR, probabilistic modeling refers to the use of a model that ranks documents in decreasing order of their evaluated probability of relevance to a user's information need. Past and present research has made use of formal theories of probability and of statistics in order to evaluate, or at least estimate, those probabilities of relevance. These attempts are to be distinguished from looser ones like, for example, the vector space model in which documents are ranked according to a measure of similarity with the query. A measure of similarity cannot be directly interpreted as a probability. In addition, similarity based models generally lack the theoretical soundness of probabilistic models. A treatment of models based on Probability Theory is beyond the scope of this entry and is reported in the relevant entries; the models presented in this section are based on the idea that IR is a process of uncertain inference.

A probabilistic formalism for describing inference relations with uncertainty is provided by *Bayesian inference networks*. Turtle and Croft [15] applied such networks to IR. Nodes represent IR entities such as documents, index terms, concepts, queries, and information needs. One can choose the number and kind of nodes one wishes to use according to how complex one wants the representation of the document collection or the information needs to be. Arcs represent probabilistic dependencies between entities and represent conditional probabilities, that is, the probability of an entity being true given the probabilities of its parents being true.

In a Bayesian inference network, the truth value of a node depends only upon the truth values of its parents. To evaluate the strength of an inference chain going from one document to the query the document node d_i is set to "true" and $P(q_k = true \mid d_i = true)$ is evaluated. This gives an estimate of $P(d_i \rightarrow q_k)$. It is possible to implement various traditional IR models on this network by introducing nodes representing Boolean

operators or by setting appropriate conditional probability evaluation functions within nodes.

Another important set of examples of logical-uncertainty models is represented by models based on *Probabilistic Datalog*. Datalog is a predicate logic that has been developed in the database field, and makes the link between the relational model and rule-based systems. *Probabilistic Datalog* is the probabilistic extension of Datalog. Probabilistic Datalog is not itself a logical IR framework, but more a platform in which logical probabilistic IR models, as well as other IR models can be expressed. One of the major assets of Probabilistic Datalog is that, since it is a generalization of the Datalog model, it can be used as standard query language for both database and IR. It can then deal with both structured data (as in database) and unstructured data (as in IR) within the same system. It also allows the uniform representation, retrieval and querying of content, fact and structural knowledge [8]. The work has been further extended via the development, implementation and evaluation of the POOL (Probabilistic Object-Oriented Logic) model, which allows the representation of inconsistency (using then a four-valued logic), the modeling of the document and query representation using the object-oriented paradigm.

A very interesting example of logical-uncertainty models and certainly the one that more reflects Van Rijsbergen's Logical Uncertainty Principle is that of models based on Logical Imaging. *Logical imaging* is an approach that defines the probability of conditional $P(d \rightarrow q)$ based on the notion of possible-worlds. In this approach the possible worlds (e.g., retrieval situation, document representation) are spanned by an accessibility relation defined in terms of similarity. The truth value of the implication $p \rightarrow q$ in a world w depends on two cases. If p is true in w, then $p \rightarrow q$ is true (false) in that world if q is also true (false) in that world. However, if p is not true in w, then the implication is evaluated in the worlds that differ minimally from w and in which p is true. The worlds in which p is true are referred to as p-worlds. The set of worlds comes with a probability distribution P, reflecting the probability of each world. The probability of a proposition p is the summation of the probability of those worlds in which p is true. The computation of the probability of $p \rightarrow q$ involves a shift of probability (the imaging process) from non-p − *worlds* to their closest p-worlds. It can be proved that: $P(p \rightarrow q) = P_p(q)$, where P_p is a new probability distribution, a "posterior probability," derived from P by imaging on p. Therefore, conditioning by imaging causes a revision of the prior probability on the possible worlds w in such a way that the posterior probability is obtained by shifting the original probabilities from non-p-worlds to p-worlds. Each non-p-world moves its probability to its closest p-world (or set of p-worlds in the case of general imaging). Bayesian conditioning, on the other hand, is obtained by cutting off all non-p-worlds and then proportionally magnifying the probabilities of the p-worlds so that the posterior probabilities still add up to one, as required by Probability Theory. The magnification is done in the same way for every p-world, thus keeping constant the ratios between the probabilities assigned to these worlds. It is therefore clear that imaging and Bayesian conditionalization yield, in general, different results. Since the transfer of probabilities is directed towards the closest p-worlds, this technique is just what it is needed to implement Van Rijsbergen's logical uncertainty principle.

Two logical-probabilistic IR models have been developed on the concept of imaging. In the first one [7], worlds model terms, and propositions model documents and queries. A term t "makes a document true" if that term belongs to that document. Imaging with respect to d gives the closest term to t that is contained in d. Of course, this is t itself if t is contained in the document. Imaging consists then of shifting the probabilities from term not contained in d to the terms contained in d (i.e., the terms that make d true). The evaluation of the relevance takes into account the semantics between terms by shifting probabilities to those (semantically) closer terms contained in the document. A second model based on imaging includes user's knowledge in the evaluation of the relevance of a document to a query [13]. In this model both documents and queries are propositions. Possible worlds represent different states of the data set, for example possible states of knowledge that can be held by users. A document d is true in a world w if the document is "consistent" (the term is used here in a broad sense) with the state of knowledge associated with that world. Worlds differ because they represent different states of knowledge and, given a metric on the world space, one can identify the closest world to w for which d is true.

Other logical-probabilistic models are based on *Semantic Information Theory*. The work on Semantic

Information Theory in IR concerns two research directions: the axiomatization of the logical principles for assigning probabilities or similar weighting functions to logical sentences and the relationship between information content of a sentence and its probability [1].

Meta-Models

Meta-models are a completely different class of models from those presented earlier. Meta-models attempts to formally study the properties and the characteristics of IR systems within a uniform logical framework. The advantage is that they make it possible to compare IR systems not only with respect to their effectiveness, but also with respect to formal properties of the underlying models.

The use of logic to formally conduct proofs for IR purposes was thoroughly investigated in [3], where a framework was proposed in which different models of IR could be theoretically expressed, formally studied and compared. The framework was developed within a logic, thus allowing formal proofs to be conducted. The framework defines the aboutness relationship, denoted $\models$, which aims at capturing the notion of information containment primary to IR. Given two objects a and b, $a \models b$ means that object a is about object b. Axioms are defined that represent possible properties of IR systems. Examples of simple axioms include reflexivity, symmetry, and transitivity, but more complex ones have also been identified, like for example weakening and monotonicity, borrowed from non-monotonic reasoning. This research has led to the theoretical comparison of IR models. The IR models are mapped down into a logic-based framework and they are compared by looking at the particular aboutness properties they each embody. Through this research, IR has gained a clearer understanding of what aboutness is, and what properties are desirable and not desirable.

Due to space limitations this entry cannot even briefly report here on other logical models of IR. A good survey of a number of these models can be found in [5]. An analysis of the strengths and limitations of logical modelling of IR can be found in [9,14]. Finally, many of the models briefly introduced here are described in details in [6].

Key Applications

The main application area of the models presented in this section is Information Retrieval.

Future Directions

This entry reviewed a number of approaches to logical modelling of IR. So far, no consensus has been reached regarding what the best approach is due to the small number of operational systems based on these approaches. It is hoped that further investigations into various logic-based frameworks accompanied by more evaluation might lead to a unified information-based model theory for expressing the semantics of information retrieval. Such a theory will enable to predict the behavior of IR systems, compare them and prove properties about them. This would be a major strength of logical models of IR.

Cross-references

- ▶ Fuzzy Logic
- ▶ Probability Theory
- ▶ Relevance

Recommended Reading

1. Amati G. and van Rijsbergen C.J. Semantic information retrieval. In Information Retrieval: Uncertainty and Logics, F. Crestani, M. Lalmas, C.J. van Rijsbergen (eds.). Kluwer, Norwell, MA, USA, 1998, pp. 189–220.
2. Bordogna G. and Pasi G. A fuzzy linguistic approach generalizing boolean information retrieval: a model and its evaluation. J. Am. Soc. Inf. Sci., 44(2):70–82, 1993.
3. Bruza P.D. and Huibers T.W.C. Investigating aboutness axioms using information fields. In Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 112–121.
4. Cooper W.S. A definition of relevance for information retrieval. Inf. Storage Retr., 7:19–37, 1971.
5. Crestani F. and Lalmas M. Logic and uncertainty in information retrieval. In Lectures on Information Retrieval, M. Agosti, F. Crestani, G. Pasi (eds.). LNCS, vol. 1980. Springer, Heidelberg, Germany, 2001, pp. 182–210.
6. Crestani F., Lalmas M., and van Rijsbergen C.J. (eds.). Information Retrieval, Uncertainty and Logics: Advanced Models for the Representation and Retrieval of Information. Kluwer, MA, USA, 1998.
7. Crestani F. and van Rijsbergen C.J. A study of probability kinematics in information retrieval. ACM Trans. Inf. Syst., 16(3):225–255, 1998.
8. Fuhr N. Probabilistic datalog – a logic for powerful retrieval methods. In Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 282–290.
9. Lalmas M. Logical models in information retrieval: introduction and overview. Inf. Process. Manage., 34(1):19–33, 1998.
10. Losada D.E. and Barreiro A. Using a belief revision operator for document ranking in extended boolean model. In Proc.

ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 66–73.
11. Meghini C., Sebastiani F., Straccia U., and Thanos C. A model of information retrieval based on a terminological logic. In Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 298–307.
12. Nie J.Y. An information retrieval model based on modal logic. Inf. Process. Manage., 25(5):477–491, 1989.
13. Nie J.Y. Lepage F., and Brisebois M. Information retrieval as counterfactuals. Comput. J., 38(8):643–657, 1995.
14. Sebastiani F. Trends in ... a critical review: on the role of logic in information retrieval. Inf. Process. Manage., 34 (1):1–18, 1998.
15. Turtle H.R. and Croft W.B. Evaluation of an inference network-based retrieval model. ACM Trans. Inf. Syst., 9(3):187–222, July 1991.
16. van Rijsbergen C.J. A new theoretical framework for information retrieval. In Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1986, pp. 194–200.
17. van Rijsbergen C.J. and Lalmas M. An information calculus for information retrieval. J. Am. Soc. Inf. Sci., 47(5):385–398, 1996.

Logical Query Processing and Optimization

► Query Processing in Deductive Databases

Logical Schema Design

► Logical Database Design: From Conceptual to Logical Schema

Logical Story Unit Segmentation

► Video Segmentation

Logical Structure

Thijs Westerveld
Teezir Search Solutions, Ede, The Netherlands

Definition

Logical structure refers to the way information in a document is organized; it defines the hierarchy of information and the relation between different parts of the document. Logical structure indicates how a document is built, as opposed to what a document contains.

Key Points

Logical structure is mainly used in the context of XML document to distinguish the logical organization of the content from its physical organization, to distinguish the flow of content from the documents layout and from the presentation of the document. In XML documents, text, images and metadata can be organized in a meaningful, logical manner independently of the document's layout when presented to a user. This contrasts with HTML documents in which logical organization and layout necessarily are the same. The logical structure of a document is of particular interest in structured document retrieval, where it may provide knowledge regarding the organization of information which may lead to better identification of relevant document parts given a user's request.

Cross-references

► Processing Structural Constraints
► Structured Text Models
► XML Retrieval

Logical Time

► Valid Time

Logical Unit Number

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonyms

LUN; Volume

Definition

LUN is a SCSI protocol term that refers to a logically contiguous piece of storage. This usually corresponds to a volume in a storage array or an individual disk drive on a parallel SCSI bus. The storage volume

identifier gets mapped to a LUN by the host operating system and they are usually different.

Key Points

A volume on a storage controller can be assigned different LUN numbers on different hosts. A volume on a storage controller can be partitioned into multiple volumes, and thus, LUNs by storage virtualization software. The storage virtualization software can reside on hosts or network virtualization boxes.

Cross-references

- ▶ LUN Mapping
- ▶ Volume

Logical Unit Number Mapping

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonyms

LUN mapping

Definition

This is the process by which the host operating system assigns a LUN value to a particular storage volume. LUN Mapping is typically used in cases where the higher level applications require specific LUN numbers for specific storage devices. When there are multiple paths between a SCSI initiator and SCSI target, multi-pathing software is used to properly map target volumes via both the paths. Storage controllers provide access control mechanisms (LUN Masking) that can control how initiators access target storage volumes.

Key Points

At host, LUN mapping is performed by operating system software. If one wants to properly manage multiple paths to a storage volume from a host, then one needs to install a multi-pathing driver on the host. LUN mapping is also performed by virtualization software that is present in network virtualization boxes. These boxes perform mapping between virtual volumes that are exported to hosts and the physical volumes residing on the storage controllers.

Cross-references

- ▶ LUN
- ▶ Multi-Pathing
- ▶ Volume

Logical Volume

- ▶ Volume

Logical Volume Manager

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Virtual disk manager; Volume set manager; LVM

Definition

Logical Volume Manager (LVM) is a kind of storage virtualization. LVM collects one or more disk drives or partitions, creating a storage pool in a host. Furthermore, LVM provides applications with logical volumes consisting of multiple chunks of disk drives or partitions.

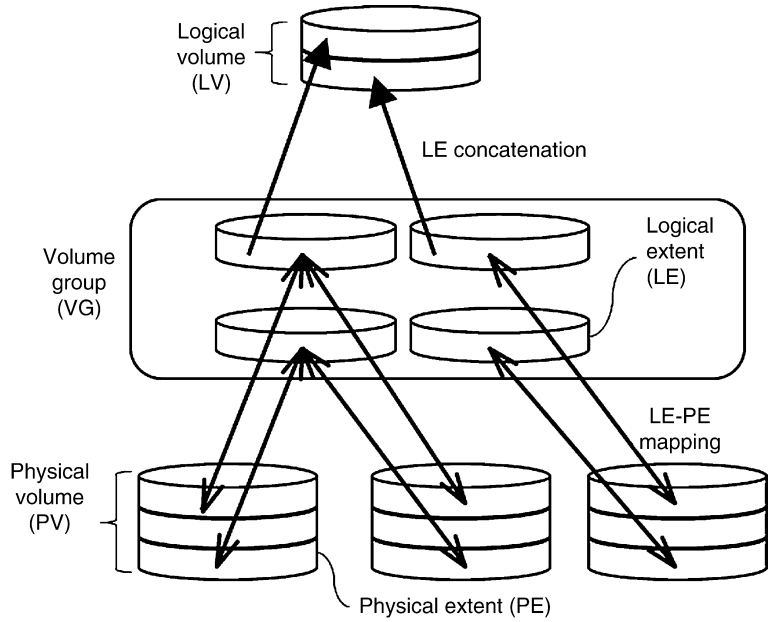
Key Points

Figure 1 shows volume manager implementation. LVM implementations often start with physical volume (PVs), which can be disk drives or partitions. PVs are spilt into chunks called physical extents (PEs). Some LVM implementations have PEs of a uniform size; others have variable-sized PEs that can be split and merged at will.

A logical extent (LE) consists of one or more PEs. For example, users can create an LE which is simply mapped to an PE, or create a striped LE which consists of multiple PEs with a conventional RAID technique. LEs are pooled into a volume group (VG).

A logical volume (LV) consists of one or more LEs in a VG. Users also can create an LV which is simply mapped to an LE, or create a striped LV which consists of multiple LEs with a conventional RAID technique.

In some implementations, LVs can be grown or shrunken by concentrating more LEs from, or returning them to, a VG. Some volume managers allow LVs



Logical Volume Manager. Figure 1. Implementation Architecture.

to be resized in either direction while online. These features help users to create flexible system configurations easily.

Cross-references

- ▶ Disk
- ▶ Redundant Arrays of Independent Disks
- ▶ Storage Management

Logical Window

- ▶ Windows

Log-Linear Regression

Jialie Shen
Singapore Management University, Singapore, Singapore

Definition

In statistic, log-linear regression is a powerful regression technique that models relationship between a dependent variable or regressand Y, explanatory variable or regressor $X = \{x_1,...,x_I\}$ and a random term ε by fitting a log-linear model,

$$\ln Y = \alpha_0 + \alpha_1 \ln x_1 + \alpha_2 \ln x_2 + ... + \alpha_I \ln x_I + \varepsilon$$

where α_0 is the constant term, the α_i s are the respective parameters of independent variables, and I is the number of parameters to be estimated in the log-linear regression.

Key Points

The goal of log-linear regression is to explore effect a set of covariance $X = \{x_1,...,x_I\}$ on the expected rate [1]. It assumes that a linear relationship exists between the log of the regressand Y and the regressor $X = \{x_1,...,x_I\}$. The log-linear regression is appropriate for categorical variables.

When applied to continuous variables, discretization is an essential step for preprocessing. There are two basic steps to using log-linear regression: (i) determining how many factors to be considered and sets of attributes related to each factor and (ii) estimating the numerical values of the parameters.

Cross-references

- ▶ Data Reduction
- ▶ Generalized Linear Models
- ▶ Linear Regression

Recommended Reading

1. McCullagh P., Nelder J. 1Generalized Linear Models, CHAPMAN & HALL/CRC, London, 1989.

Long Running Queries

► Continuous Queries in Sensor Networks

Longitudinal Health Record

► Electronic Health Records (EHR)

Looking Over/Through

► Browsing in Digital Libraries

Loop

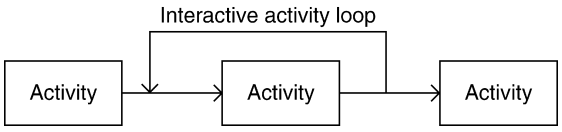
Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Iteration; Workflow loop; While loop

Definition

A workflow activity cycle involving the repetitive execution of one (or more) workflow activity(s) until a condition is met.



Key Points

A Loop is a control flow statement that allows an activity or activity block to be executed repeatedly, rather than proceeding to the next activity, until a specific condition has been met. Following the execution of the activity (or last activity in the block) the condition of the loop is evaluated.

Cross-references

► Activity
► Process Life Cycle
► Workflow Management and Workflow Management System

Loop Join

► Nested Loop Join

Loose Coupling

Serguei Mankovskii
CA Labs, CA, Inc., Thornhill, ON, Canada

Synonyms

Weak coupling; Low coupling

Definition

Word coupling refers to a notion of interdependence between components of software systems. Word loose refers to a mode of coupling where components possess significant degree of autonomy.

Key Points

Term loose coupling consists of two opposite notions of dependency and autonomy. It initially emerged out of study of educational organizations. Organizational scientist Karl Weick noticed that some organizations can be mechanistic (coupled) and organic (loose) at the same time. Loosely coupled organizations are under influence of forces or conditions that hold them together, but at the same time they are acting with the high degree of independence.

In software engineering, loose coupling has similar notion to organizational theory. A loosely coupled system usually consists of a framework or architecture linking together otherwise independent components. The components are usually highly cohesive and have well defined responsibilities. Cohesive components allow for creation of stable interfaces exposing less implementation details through the component interface. Loosely coupled system uses stable interfaces of the components to hold the system together.

As a design, goal loose coupling is desirable when system components need to be developed or used independently of each other and when they need to change over time. This goal can be achieved by deliberate de-coupling of system components. De-coupling leads to a more reliable system design and reduces maintenance costs over the life time of system and its components.

Software components developed for loose coupling can be readily re-purposed for use in a broad class of systems leading to higher re-use. They can also be used at the same time by a broad class of systems through standards-based interfaces. In this case they are often called services. Service Oriented Architecture is a special case of software systems built around of loosely coupled services.

Cross-references

- ▶ Cohesion
- ▶ Coupling and De-Coupling
- ▶ Interface
- ▶ Re-Use
- ▶ Service Oriented Architecture

Recommended Reading

1. Wieck K.E. Educational organisations as loosely coupled systems. Adm. Sci. Q., 21:1–19, 1976.

Lossless Data Compression

▶ Text Compression

LoT-RBAC

▶ GEO-RBAC Model/ExternalRef>

Low Coupling

▶ Loose Coupling

Lp Distances

▶ Frequency Moments

Lp Norms

▶ Frequency Moments

LSID

▶ Semantic Data Integration for Life Science Entities

LUN

▶ Logical Unit Number (LUN)

LUN Mapping

▶ Logical Unit Number Mapping

LUN Masking

▶ Storage Security

LVM

▶ Logical Volume Manager