World Wide Web 1 (1998) 241–255

Query processing with quality control in the World Wide Web

Ying Chen [a], Qiang Zhu [b] and Nengbin Wang [a]

[a] *Department of Computer Science and Engineering, Southeast University, Nanjing 210096, P.R. China*
E-mail: {ychen, nbwang}@seu.edu.cn.

[b] *Department of Computer and Information Science, The University of Michigan – Dearborn, Dearborn, MI 48128, USA*
E-mail: qzhu@umich.edu.

Recent research on integrating database and World Wide Web (WWW) technologies has changed the navigation approach to searching information in the Web. People now can issue queries via a simple query interface or a database-like query language to retrieve information from semistructured WWW data sources. However, the quality of query processing in the WWW is still low due to many factors such as unpredictable response time, irrelevant results, and out-of-date data. Such low-quality query processing is intolerable to either users or service providers. In this paper, we present a quality-controlled query processing method in the WWW. Quality parameters that users can specify with their queries are introduced. Distance functions that are used to evaluate the goodness of query quality parameters are defined. A query processing model with quality control is introduced. A quality control protocol in query processing is presented. Quality-controlled query scheduling algorithms including admission scheduling, promotion/demotion scheduling and execution scheduling are proposed. Other relevant issues such as query classification, system parameter estimation, and query queue management are also discussed. Query processing with quality control is a promising way to solve the uncertain and low-quality query processing problems in the WWW.

1. Introduction

The World Wide Web (WWW or simply known as Web) has become the leading information service in the Internet. At present there are a large number of applications and services based on the Web, such as electronic publication, electronic commerce, distance learning, remote medicine, video on demand (VOD), etc. The ultimate goal of the Web is to become the platform of personal information systems (PIS) as well as of enterprises [Berners-Lee 1996].

Since an increasing amount of information is stored in the Web servers that are connected by anchors in HTML (HyperText Markup Language) documents, people who browse information in the Web often get lost in the hyperspace. The severe limitation of browsing as a search technique has become a main obstacle to prevent users from mining data in which they are interested. Although there are many resources in the Web, people may be unable to find or use them effectively. To facilitate users to search information in the Web, many search engines have been deployed on the Web, such as OpenText, AltaVista, Lycos, and WISE, as the major tools for resource discovery [Winder 1997]. A typical search engine provides users with a query interface and employs a piece of software called Robot which works continuously in the background to roam around Web servers, fetch documents from them, analyze the gathered documents, and put them in an index database. A user specifies and submits a query by filling the query condition in a form. After the query is processed, the search engine returns a list of URLs (Uniform Resource Locator) to the user. Recently, people have studied the issues on how to support database-like queries in the Web [Konopnicki and Shmueli 1995; Mendelzon *et al.* 1996; Chen *et al.* 1998]. A user can issue an SQL-like query to retrieve information from the Web. However, the types of queries that can be expressed are still very restrictive. In addition, the underlying search engine needs to provide a structured view of the semistructured data in the Web [Ashish and Knoblock 1997; Hammer *et al.* 1997].

Due to the nature of Web (such as huge data volume, distributed data, frequent changing, network time delay, etc.), query results are usually inaccurate, compared with the query results in a traditional database system. Furthermore, the limitations of the power of existing query language/interface (most of them only allow a simple boolean expression in a query like the keyword search in an information retrieval system) reduce the effectiveness of query specification. These factors lead to low-quality queries that have the following undesirable features:

- Unpredictable response/turnaround time. Since there are a huge number of links in the Web servers and query results are usually inaccurate, a user usually has no idea about how long his query will take. It creates difficulties for a user to effectively utilize his/her time.
- A large amount of irrelevant results. A Web search engine often returns a lot of irrelevant results to users. Browsing irrelevant query results (Web trash) wastes the users' precious time as well as network resources.
- Out-of-date results. Due to frequent changing of Web documents, a user may get old information from the local replicas in a search engine. In addition, many users are interested in new information that has just appeared in the Web. These users are often annoyed by a large amount of out-of-date results.

From the viewpoint of service providers, performing low-quality queries is abusing the precious system/network resources. From the viewpoint of users who pay the Internet service providers for services, providing low-quality query processing is wasting their money. It is clear that that low-quality query services are intolerable to either users or providers.

In this paper, we introduce a quality-controlled query processing method in the Web environment. Users can specify their queries together with some requirements for a set of quality parameters. A query quality parameter reflects a user's subjective expectation by using a measurable value, such as query response time, query result size, etc. A query server (i.e., a search engine) processes queries with quality parameters under the guidance of a quality controller to meet the users' quality requirements within the limitations of system resources such as memory size and secondary storage space, CPU speed and I/O bandwidth, and network load, etc. As mentioned before, due to the nature of the Web, obtaining accurate query results is often unrealistic. Different requirements for query quality parameters may generate different results that reflect different users' preferences. Note that since the system resources are limited, it is sometimes impossible to meet the quality requirements for all queries. In this case, a few queries have to be sacrificed in order for other queries to meet their quality requirements. However, the partial result of a discarded query together with its reached values of quality parameters will be returned to the user. Our goal is to meet the quality requirements for as many queries as possible rather than to achieve very high quality processing for only a few queries.

A number of techniques for Web queries have been studied in the literature, including query languages/interfaces, organizations of index databases, query processing, ranking algorithms, etc. A simple form-based interface is supported by most search engines in the Web [Winder 1997]. Some work has been done on database-like query languages for the Web [Konopnicki and Shmueli 1995; Mendelzon *et al.* 1996; Chen *et al.* 1998]. Konopnick and Shmueli [1995] introduced an SQL-like language W3QS, which allows both structure queries and content queries and supports extensibility and interfacing to external user-written programs and Unix utilities. Mendelzon *et al.* [1996] discussed another SQL-like language WebSQL and its implementation. The formal semantics of WebSQL using a calculus based on a "virtual graph" model were given in the paper. A new theory of query cost based on "query locality" was introduced. Chen *et al.* [1997] discussed query processing and optimization for a query language WWWQL. Some new concepts, such as link coupling degree between anchors and link aggregation degree, were introduced. The approach employed by Fiebig *et al.* [1997] is to extend the relational algebra so that it can be used to process queries in the Web. Levy *et al.* [1996] discussed the query plan algorithms used in Information Manifold System, which are applicable to data sources in the Web as well as other information sources. The database organizations that are widely used in search engines are full-text indexing. Ranking is another technique for sorting query results via assigning ranks to them according to their relevancy to the user queries. Four keyword-based search and ranking algorithms, including the well-known TF×IDF vector space model, were introduced in [Yuwono and Lee 1996]. Fiebig *et al.* [1997] and Chen *et al.* [1997] also discussed some ranking methods.

Although there were some discussions on how to manage information for quality control in database applications [Gosinski *et al.* 1997; Kim *et al.* 1995], to our knowledge, no work has been done on incorporating quality control into query processing in the Web.

The rest of this paper is organized as follows. Section 2 introduces the concepts of quality control for query processing in the Web. Section 3 discusses a quality-controlled query processing model. Section 4 presents algorithms for quality-controlled query processing in the Web, including estimating system parameters and query scheduling. Section 5 concludes the paper.

2. Query quality control

To perform query processing with quality control, we need to identify suitable quality parameters, introduce their assessment methods, and describe the way to specify quality-controlled queries. These are the issues to be discussed in this section.

2.1. Query quality parameters

Query quality is the degree of satisfaction by a user with the query result and query processing according to his/her requirements. The more satisfied the user is, the higher the query quality. The qualitative/subjective nature of query quality makes it difficult to take query quality into consideration during query processing. To incorporate quality control into query processing, query quality must be measured quantitatively. Hence we introduce a set of parameters whose values reflect different indexes related to query quality (see table 1).

The term 'item' used in table 1 is defined as one URL of a relevant document because URLs are the identifiers of resources in the Web. The query granularity (*GR*) is a subset of TAGSET that is the set of all HTML tags [Berners-Lee and Conolly 1995]. An *GR* value specifies a minimum set of HTML tags that query processing has to search. An *GR* value $\{\langle TITLE\rangle, \langle B\rangle, \langle HREF\rangle\}$, for example, means to search the text at least appearing in the title, bold parts, and anchors of each HTML document. An *GR* value $\{\langle HTML\rangle\}$, as another example, means to search the whole document. The accuracy of result (*AC*) is defined as

$$AC = NC/QR,$$

Table 1
Typical query quality parameters in the Web.

Parameter	Description	Example value
RT	Query response time – from the time the query is submitted to the time the first piece of data appears.	30 seconds
TT	Query turnaround time – from the time the query is submitted to the time the query is completed (i.e., all results are returned).	2 minutes
GR	Query granularity – query processing search scope in a Web document, e.g., title, section head, text in bold type, etc.	{⟨*TITLE*⟩, ⟨*B*⟩, ⟨*HREF*⟩}
AC	Accuracy of result – percentage of items satisfying the query qualification condition.	0.6
RL	Relevancy of result – percentage of items that may not satisfy the query qualification condition but are relevant.	0.8
SZ	Size of result – number of items in a query result.	20 items
FR	Fresh degree of result – the duration of the oldest retrieved item since its existence. A user may not be interested in very old items.	15 days
NS	Number of sites covered – number of Web servers accessed.	100 sites
ND	Number of Web documents accessed.	1000 documents
RR	Redundancy rate – percentage of duplicate items.	0.05

where QR is the number of all items in the query result and NC is the number of correct items in the query result. As for the relevancy of result (RL), it is defined as

$$RL = \sum_{i=1}^{NC} rank(item_i)/QR,$$

where $0 \leqslant rank(item_i) \leqslant 1$ is the relevancy rank of $item_i$. If every item has a relevancy rank of either 1 (totally relevant, i.e., correct) or 0 (totally irrelevant, i.e., incorrect), then RL becomes AC. In other words, AC is a special case of RL. FR indicates the longest duration of existence for items that are allowed in the query result.

For the values of some query quality parameters (such as response time), the smaller, the better. While for the values of some other query quality parameters (such as accuracy of result), the bigger, the better. A user can specify a minimum requirement (maximum/minimum value) for each quality parameter when he/she issue a query. A query processor can check if the processing of a query meets the quality requirements by examining the values of quality parameters. If any quality parameter of a query has a value below the minimum required value or beyond the maximum tolerated value when the query processing is finished, the quality checking for the query fails.

Note that the quality-controlled query processing algorithms introduced this paper can be easily extended to incorporate additional query quality parameters if any. On the other hand, a user may not be interested in every quality parameter in table 1. In the latter case, a NULL value is used for the parameter. Our quality control algorithms will not check the quality parameters that a user is not interested in. In the extreme case, if a user does not specify any quality parameters, the quality-controlled query processing is reduced to the traditional query processing.

The query quality parameter values specified by a user are called *required quality parameter values*. The query quality parameter values achieved at a moment in time during query processing are called *current quality parameter values*. The query quality parameter values reached after query processing is finished are called *finished quality parameter values*. If the finished quality parameter values are equal to or better (smaller or greater depending on the case) than the required quality parameter values for a given query, the query is processed successfully under quality control.

To evaluate the relative goodness of two query quality parameter values, we employ a distance function which is defined as follows:

Definition 1. Let $Dom(P)$ be the domain of query quality parameter P and NULL $\notin Dom(P)$. Let $p_1, p_2, p_3 \in Dom(P)$. A function δ_P is a distance function for P if it has following properties:

(1) δ_P is a function from $Dom(P) \times Dom(P)$ to $\mathbb{R}$, where $\mathbb{R}$ is the set of real numbers;

(2) $\delta_P(p_1, p_2) > 0$ indicates that parameter value p_1 is better than parameter value p_2;

(3) $\delta_P(p_1, p_2) = 0$ if and only if $p_1 = p_2$;

(4) $\delta_P(p_1, p_2) > 0$ if and only if $\delta_P(p_2, p_1) < 0$;

(5) if $\delta_P(p_1, p_2) > 0$ and $\delta_P(p_2, p_3) > 0$, then $\delta_P(p_1, p_3) > 0$.

Property (4) implies that if p_1 is better than p_2, then p_2 is worse than p_1. Property (5) implies that if p_1 is better than p_2 which is better than p_3, then p_1 is better than p_3. Both are desirable properties of a distance function for quality control processing.

For each query quality parameter, we need to choose a distance function satisfying properties (1)–(5). A non-negative value of a distance function for two parameter values indicates that the first parameter value is at least as good as the second one.

The following are feasible distance functions for the query quality parameters in table 1:

(a) $\delta_P(p_1, p_2) = p_2 - p_1$, where $p_1, p_2 \in Dom(P)$ and $P \in \{RT, TT, RR, FR\}$; which implies that, the smaller the parameter value is, the higher the quality.

(b) $\delta_P(p_1, p_2) = p_1 - p_2$, where $p_1, p_2 \in Dom(P)$ and $P \in \{AC, RL, SZ, NS, ND\}$; which implies that, the larger the parameter value is, the higher the quality.

(c) For parameter *GR*, we define

$$\delta_{GR}(p_1, p_2) = \begin{cases} \|p_1\| - \|p_2\|, & \text{if } p_2 \subseteq p_1, \\ -1, & \text{otherwise,} \end{cases}$$

where $p_i \in Dom(GR)$ $(i = 1, 2)$ and $\|p_i\|$ is the cardinality of set p_i.[1]

Note that a distance function is not defined on a NULL parameter value since NULL implies that the user does not require the system to check the relevant query quality parameter.

In the rest of the paper, let $P_r(Q)$, $P_c(Q)$, and $P_f(Q)$ denote the required value, current value, and finished value of parameter P for query Q, respectively.

2.2. Query quality parameter vectors and their distance function

A distance function in definition 1 is for a single quality parameter. However, a user usually specifies a set of quality parameters for a query. To compare two sets of values of quality parameters, we need to introduce an aggregate distance function that allows us to compare quality parameters collectively.

Definition 2. Let $P_1, P_2, \ldots, P_n$ be a set of query quality parameters and NULL $\notin Dom(P_i)$ $(i = 1, 2, \ldots, n)$. Let $\vec{P} = (P_1, P_2, \ldots, P_n)$ be the parameter vector whose domain $Dom(\vec{P}) = Dom(P_1) \times Dom(P_2) \times \cdots \times Dom(P_n)$. Let $\vec{p}_i = (p_{i1}, p_{i2}, \ldots, p_{in}) \in Dom(\vec{P})$ $(i = 1, 2)$. A function Δ is a distance function for $\vec{P}$ if it has following properties:

(1) Δ is a function from $Dom(\vec{P}) \times Dom(\vec{P})$ to $\mathbb{R}$, where $\mathbb{R}$ is the set of real numbers;

(2) $\Delta(\vec{p}_1, \vec{p}_2) = \Psi(\delta_{P_1}(p_{11}, p_{21}), \delta_{P_2}(p_{12}, p_{22}), \ldots, \delta_{P_n}(p_{1n}, p_{2n}))$, where Ψ is a function from $\mathbb{R}^n$ to $\mathbb{R}$, and δ_{P_k} is a distance function for query quality parameter P_k $(k = 1, 2, \ldots, n)$;

(3) $\Delta(\vec{p}_1, \vec{p}_2) > 0$ if and only if there exists j $(1 \leqslant j \leqslant n)$ such that $\delta_{P_j}(p_{1j}, p_{2j}) > 0$ and $\delta_{P_i}(p_{i1}, p_{i2}) \geqslant 0$ $(i = 1, 2, \ldots, j-1, j+1, \ldots, n)$;

(4) $\Delta(\vec{p}_1, \vec{p}_2) = 0$ if and only if $\vec{p}_1 = \vec{p}_2$;

(5) $\Delta(\vec{p}_1, \vec{p}_2) > 0$ if and only if $\Delta(\vec{p}_1, \vec{p}_2) < 0$.

The following proposition states that the transitive property is implied by the properties given in definition 2.

[1] In HTML, some tags may imply other tags, e.g., ⟨*HTML*⟩ implies all other tags. A set used here means the set after expanding the relevant super tags.

Proposition 1. The distance function Δ defined in definition 2 satisfies the following property: if $\Delta(\vec{p}_1, \vec{p}_2) > 0$ and $\Delta(\vec{p}_2, \vec{p}_3) > 0$, then $\Delta(\vec{p}_1, \vec{p}_3) > 0$.

Proof. If $\Delta(\vec{p}_1, \vec{p}_2) > 0$ and $\Delta(\vec{p}_2, \vec{p}_3) > 0$, by property (3) in definition 2, there exists j $(1 \leqslant j \leqslant n)$ such that $\delta_{P_j}(p_{1j}, p_{2j}) > 0$ and $\delta_{P_i}(p_{i1}, p_{i2}) \geqslant 0$ $(i = 1, 2, \ldots, j-1, j+1, \ldots, n)$ and $\delta_{P_i}(p_{i2}, p_{i3}) \geqslant 0$ $(i = 1, 2, \ldots, n)$. We have $\delta_{P_j}(p_{1j}, p_{3j}) > 0$ no matter $\delta_{P_j}(p_{2j}, p_{3j}) = 0$ (applying property (3) in definition 1) or $\delta_{P_j}(p_{2j}, p_{3j}) > 0$ (applying property (5) in definition 1). Similarly, we have $\delta_{P_i}(p_{i1}, p_{i3}) \geqslant 0$ $(i = 1, 2, \ldots, j-1, j+1, \ldots, n)$. Therefore, $\Delta(\vec{p}_1, \vec{p}_3) > 0$. □

Let $\vec{P}_r(Q)$, $\vec{P}_c(Q)$, and $\vec{P}_f(Q)$ denote the vector of required quality parameter values, the vector of current quality parameter values, and the vector of finished quality parameter values for query Q, respectively. Assume that $\Delta(\vec{P}_f(Q), \vec{P}_r(Q)) = -\infty$ if $\vec{P}_f(Q)$ is undefined (i.e., values are missing for some component parameters). We determine the quality of query processing according to the following rules:

- If $\Delta(\vec{P}_f(Q), \vec{P}_r(Q)) \geqslant 0$, the processing of query Q meets the quality requirements specified by the user.
- If $\Delta(\vec{P}_f(Q), \vec{P}_r(Q)) = -\infty$, the query processing has not been finished or has been aborted.

It is clear that if one of finished quality parameter values in $\vec{P}_f(Q)$ is less than its corresponding required quality parameter value in $\vec{P}_r(Q)$, $\Delta(\vec{P}_f(Q), \vec{P}_r(Q))$ is negative. Therefore, distance function Δ guarantees a strict quality evaluation.[2] As mentioned before, the result of a query in the Web is usually inaccurate. We consider the result of query Q to be correct if $\Delta(\vec{P}_f(Q), \vec{P}_r(Q)) \geqslant 0$. The transitivity of function Δ (proposition 1) allows query processing with quality control to make $\Delta(\vec{P}_f(Q), \vec{P}_r(Q))$ as large as possible (i.e., getting better and better quality) within the limitations of system resources.

Any function satisfying properties (1)–(5) in definition 2 can be used as a distance function for parameter vectors. In this paper, we adopt the following distance function for parameter vectors:

$$\Delta(\vec{p}_1, \vec{p}_2) = \Psi(d_1, d_2, \ldots, d_n) = \begin{cases} \sum_{i=1}^{n} K_i d_i, & \text{if } d_i \geqslant 0 \text{ for all } i, \\ \sum_{d_i < 0} K_i d_i, & \text{if there exists } i \text{ such that } d_i < 0, \end{cases} \quad (1)$$

where $d_i = \delta_i(p_{1i}, p_{2i})$, and $K_i \geqslant 0$ is a weight for d_i.

K_i in formula (1) reflects the relative importance of P_i in the quality parameter vector and also adjusts the difference of measuring units of different quality parameters. To determine K_i, the following three approaches can be utilized:

[2] Strict quality evaluation is typical in quality control. If a user wants to relax the strict quality evaluation, our quality controller can be easily modified to allow a certain percent of quality parameters to violate the specified quality requirements to a certain degree, or the user specifies more relaxed quality parameter values.

(a) *Static approach.* K_i is determined by the system in advance. The approach is simple. The weights may not quite reflect the users' requirements.

(b) *Regression with sampling.* Consider the following sampling data from some users:

$$\begin{aligned}&\left(d_1^{(j)}, d_2^{(j)}, \ldots, d_n^{(j)}\right) \to Y^{(j)},\\ &0 \leqslant Y^{(j)} \leqslant 100, \quad j = 1, 2, \ldots, m,\\ &d_i^{(j)} \geqslant 0, \quad i = 1, 2, \ldots, n,\end{aligned}$$

where $Y^{(j)}$ denote the ranks evaluated by the users for m sets of distances $(d_1^{(j)}, d_2^{(j)}, \ldots, d_n^{(j)})$ for quality parameters. We can get the following equations using the sampling data:

$$Y^{(j)} = K_1 d_1^{(j)} + K_2 d_2^{(j)} + \cdots + K_n d_n^{(j)} + \mu^{(j)}, \quad (2)$$

where $\mu^{(j)}$ denotes the random error term. It is usually assumed that $\mu^{(j)}$ follows the normal distribution and its expected value is 0. Hence we view (2) as a regression equation and estimate the weights K_i (regression coefficients) by using the ranks from the users together with the sets of given distances for quality parameters. The more data we sample, the better estimates we can get for K_i.

(c) *Self-adaptive learning.* The linear self-adaptive neural network is adopted in this approach. The main idea is to continuously adjust K_i according to some users' feedback until K_i converges to a tolerated range. More specifically, $(K_1^{(0)}, K_2^{(0)}, \ldots, K_n^{(0)})$ are initially determined by using the static approach. Assume the system has computed rank $Y^{(j)} = \sum_{i=1}^{n} K_i^{(j-1)} d_i^j$ at step j $(j \geqslant 1)$. Let $R^{(j)}$ denote the users' feedback for rank $Y^{(j)}$. According to the least means sum (LMS) principle of Widrow–Hoff learning rule, weight K_i is adjusted as follows:

$$\begin{aligned}&\Delta K_i^{(j)} = \alpha \cdot d_i^{(j)}\left(R^{(j)} - Y^{(j)}\right),\\ &K_i^{(j)} = K_i^{(j-1)} + \Delta K_i^{(j)},\\ &\quad i = 1, 2, \ldots, n, \quad j \geqslant 1,\end{aligned}$$

until $\Delta K_i^{(j)} < E_i$, where α is the ratio of learning $(0 < \alpha < 1)$, and E_i is a tolerated range. This approach is usually better than the previous two approaches since unlimited feedback can be obtained from users and dynamic changes are made according to users' expectation at different times. However, a feedback mechanism to receive users' ranks is required.

To further improve the estimates of weights, we can derive different sets of K_i's for different query classes based on the classification to be discussed in the following subsection.

2.3. Classification of Web queries

Different people search the Web for different purposes. Some people only want to get a quick reference about a topic via a search engine. A few URLs are sufficient for them to navigate the relevant data sources. Some people may want to obtain a large number of URLs, even abstract texts extracted from the Web pages. Some other people are interested in checking the most recent information in the Web. Based on the characteristics of Web queries, we divide them into four classes as shown in table 2.

Table 2 also shows the degrees of requirements for major quality parameters by each query class. As mentioned in the last subsection, the degrees of requirements help to determine proper weights K_i's in the distance function (1). If the user's requirement for a quality parameter is very low, he/she may not even specify a value for it (i.e., NULL). In this case, the system will not check this parameter, which means that any finished value for the parameter is acceptable. We assume that queries discussed in this paper fall into one of the four classes although our technique can be extended to handle more query classes.

2.4. Objectives of query quality control

The objectives of query quality control are to:

(a) meet the requirements for query quality parameters specified by a user within the limitations of system resources,

(b) maximize the system throughput,

(c) maximize the value of $\Delta(\vec{P}_f(Q), \vec{P}_r(Q))$ for a query Q if (a) and (b) are met,

(d) notify the user as soon as possible if (a) cannot be met and return partial query result (if any) with its reached values of quality parameters.

Table 2
Four query classes in the Web. L, M, and H denote low, medium, and high requirement for the relevant quality parameter, respectively.

Label	Class	Feature	Use	*RT/TT*	*SZ*	*AC*	*FR*	*NS*
$C^{(1)}$	Urgent	Short response time and relatively low accuracy	Real-time query	H	L	M	M	L
$C^{(2)}$	Precise	High accuracy	Information retrieval	L	M	H	H	L
$C^{(3)}$	Quantitative	Large quantity of result items	Data collecting	L	H	M	M	H
$C^{(4)}$	Sample	Varieties of sites accessed and relatively low quantity	Data sampling	M	L	H	M	L

Since many queries may be processed at the same time and the system resources are limited, it is sometimes unrealistic to guarantee every query processing to meet its quality requirements. Query quality control is in the statistical sense. That is, the processing of most queries will meet their quality requirements although the processing of some queries may not meet their quality requirements. A few queries may even be aborted to guarantee quality processing of many other queries.

2.5. Specification of query with quality parameters

A query Q with quality parameters can be specified as a triple $Q = \langle QueryClause, \vec{p}, C\rangle$, where *QueryClause* is a query statement; $\vec{p}$ is the vector of quality parameter values; $C \in \{C^{(1)}, C^{(2)}, C^{(3)}, C^{(4)}\}$ is a query class listed in table 2.

For example, using the terms of the Web query language WWWQL in reference [Chen *et al.* 1998], a query with quality parameters $Q = \langle$"*SELECT NODEURL WHERE 'Query', 'Quality' IN Content*", (TT = 30 s; $GR = \{\langle B\rangle, \langle H1\rangle, \ldots, \langle H6\rangle\}$; NR = 20), $C^{(1)}\rangle$ represents an urgent query to find documents containing '*Query*' and '*Quality*' in the texts embedded at least in tags $\langle B\rangle, \langle H1\rangle, \ldots, \langle H6\rangle$ and to return at least 20 URLs within 30 seconds.

3. Query processing model with quality control

How to process a query with quality parameters is the issue to be discussed in this section. A query processing model with quality control will be used to process such queries. A related query scheduling model will show how queries are scheduled within the query processing model.

3.1. Query processing model

Our query processing model with quality control consists of a Query Processor, a Robot, a Quality Controller, a Database, a Meta-Database, and an Interface Layer (see figure 1). The Robot is continuously running in the background to gather data from Web documents, build indexes for interesting keywords, and store them in the Database (DB) and Meta-Database (MetaDB). Users submit queries with quality parameters to the query processing system via an interactive interface or an application programming interface (API) in the Interface Layer. Upon receiving a user query, the Query Processor scans and parses the query for checking possible syntactic and semantic errors. The Query Processor performs query optimization to generate a good execution plan for the query, and then execute the query according to the plan. The query result will be returned to the user via the Interface Layer. If the query is aborted, partial result along with values of current quality parameters will be returned. Users can adjust their quality parameter values according to the returned values. To reduce the number of query abortions, the Query Processor and the Quality Controller can, based on the current system resource utilization, provide some recommended ranges of quality parameters for a user via the Interface Layer when he/she specifies a query. The Query Processor can process several queries simultaneously via multiple threads. The Robot may be requested to fetch and index new data when the desired information is not in the Database or out-of-date. On the other hand, the Query Processor executes queries under the control of Quality Controller. The Quality Controller, cooperating with the Query Processor, is in charge of controlling quality of query processing. For example, the Quality Controller manages query queues, schedules queries according to their quality parameter requirements and available system resources, and evaluates quality parameter values of queries. The Quality Controller uses data from the Database and MetaDB and maybe some new data retrieved by the Robot on the fly to fulfill its quality control tasks. Since the workings of the Robot, Interface Layer, and the Query Processor are not much different from the ones in a typical Web search engine or a traditional database system, the rest of the paper will focus on discussing quality control issues for the Quality Controller.

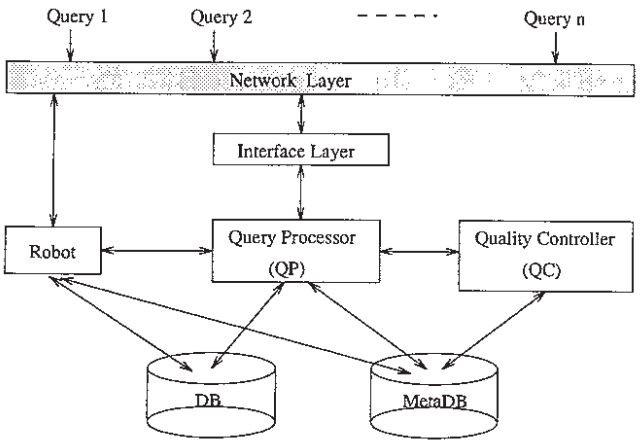

Figure 1. Query processing model with quality control.

The Database can be implemented as a relational database or an object-oriented database or a proprietary database based on full-text indexing. Due the difficulties caused by the semistructured data in the Web, most search engines such as Lycos, WebCrawel and OpenText employ the last type of databases. However, some work has been done in the literature on building a relational view or an object-oriented view for a search engine in the Web [Ashish and Knoblock 1997; Hammer *et al.* 1997]. How to organize the data in the Database is a separate issue that is not related to query quality control. Existing techniques can be utilized. Therefore, it will not be further discussed in this paper. MetaDB is used to store meta-information, such as system model parameters to be discussed in section 3.3, which will be used by the Quality Controller.

The query processing model presented in this subsection has some common features with many existing search engines. It is expected that the quality control technique developed for this model can also be incorporated into existing search engines.

3.2. Query scheduling model

There are two queues managed by the Quality Controller. One is a waiting queue QL_w which contains the queries that are waiting for a thread. The other is an active queue QL_a which contains the queries that have been allocated a thread.

The state transition diagram of a query Q is described in figure 2. There are five states in the life cycle for Q: waiting for admission (WA), waiting (W), active (A), running (R), and terminated (T). To move the query from one state to another, three types of scheduling are needed: admission scheduling, promotion/demotion (PT/DT) scheduling, and execution scheduling.

A submitted query Q starts in state WA, i.e., waiting for admission. Q may be rejected and enters state T if the system resources and load are unable to meet the query quality parameter requirements. Otherwise Q moves to state W and enters queue QL_w to wait for a chance to be promoted by the PT/DT scheduling. Once being promoted, i.e., being allocated a thread, Q moves to state A and enters queue QL_a. Q moves to state R for execution after competing for CPU with other queries under the execution scheduling. Q moves back to state A when its time slice is over. After the execution of Q is completed, it exits. Note that the number of queries in state R depends on the number of CPUs. In this paper, we assume that the system has only one CPU.

The Quality Controller may find that some queries in queue QL_w or QL_a will not be able to meet their quality requirements during scheduling. In such a case, these queries are aborted and enter state T. The partial results of the queries together with their reached values of quality parameters are returned to their users.

Furthermore, the Quality Controller may find that a query Q_1 in queue QL_a has met or almost met its quality requirements while another query Q_2 in QL_w needs immediate service, or else its quality requirements will not be met. In this case, the states of Q_1 and Q_2 are exchanged, i.e., Q_1 moves from state A to state W and Q_2 moves from state W to state A. The partial result of Q_1 is saved and will be reloaded when it moves back to state A from state W.

The queue management and scheduling algorithms will be discussed in more details in sections 4.3 and 4.4.

3.3. Model parameters

There are several model parameters that the Quality Controller and the Query Processor use to schedule and process queries with quality parameters. Query quality parameters reflect users' requirements for the quality of query processing, while the model parameters reflect the underlying system environment. However, some model parameters such as query cost may also be used by a user as query quality parameters. The main model parameters that are used in quality control are: query cost, network delay, and query queue sizes.

3.3.1. Query cost

Query cost (QC) reflects performance of the system. Although in general there are different criteria to measure query cost, execution time is the one people have used most often. We also consider execution time as query cost in this paper. If a user specifies a value for query quality parameter TT, it actually gives an upper bound for the cost of his/her query. If we have good estimates for the costs of queries, the queries that will be unable to meet their requirements of quality parameter TT can be discovered earlier before their executions are completed. Such queries can be rejected for admission or aborted earlier to allow other queries to use the system resources. In this way, we can achieve our goal to have as many queries meet their quality requirements as possible.

In general, the cost QC of a query consists of two parts: local cost QC_L, which is the cost for searching the local database; remote cost QC_N, which is the cost for fetching data from the network. In other words, the cost QC of query Q is $QC(Q) = QC_L(Q) + QC_N(Q)$. The cost QC_L is determined by the physical organization of the underlying database, the employed query processing algorithms, I/O speed, buffer sizes, etc. The cost QC_N is mainly determined by network bandwidth and delay. For a query Q involving remote data fetching, $QC_N(Q)$ is usually much higher than $QC_L(Q)$.

3.3.2. Network delay

Network delay (T_{nd}) is the time for transmitting a unit package in network, which is the most significant parameter affecting query performance in the Web. T_{nd} is mainly determined by network bandwidth, network load, and efficiency of network protocols. Network bandwidth, which is a bottleneck of Internet, and protocols are fixed when the system was built up. However, network load is varying from time to time. Clearly, T_{nd} depends largely on the dynamically changing network load. Hence, information about network load is the key to estimate T_{nd}. The estimates of this parameter are used to estimate the remote cost QC_N of a query.

3.3.3. Query queue sizes

The sizes of query queues QL_a and QL_w are constrained by system resources. If the queue sizes are too large, the Quality Controller would allow too many queries to compete for limited system resources, which would dramatically degrade the overall system performance. The quality of query processing is, therefore, very low since many queries may be unable to meet their response/turnaround time requirements. On the other hand, if the queue sizes are too small, the Quality Controller may not take full advantage of system resources to make as many queries meet their quality requirements as possible since many queries may be rejected or aborted before they are completed. A trade-off is required to determine a proper size for a queue. It appears that it is better to allow the size of a queue dynamically change according the system load. However, for

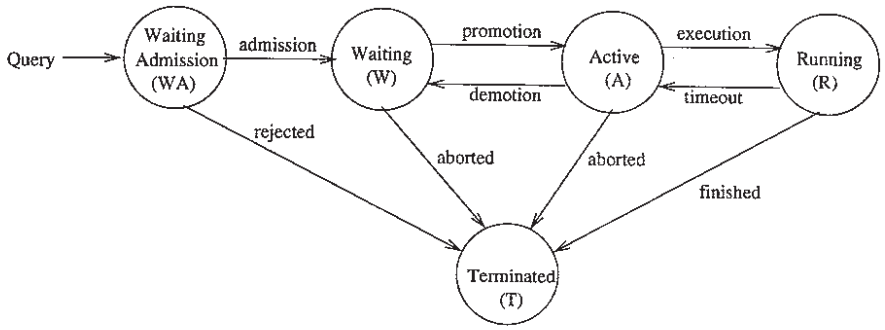

Figure 2. Query state transition diagram.

simplicity, we assume the size of a query queue is fixed in this paper. Note that the algorithms to be discussed in following section can also be used for the queues with variable sizes.

4. Quality control algorithms for query processing

To realize the above query processing model with quality control, in this section, we introduce the methods for estimating the model parameters, the protocol for controlling quality in query processing, and the algorithms for managing queues as well as scheduling queries.

4.1. Estimating model parameters

Estimating the parameters in section 3.3 accurately is unrealistic due to the uncertain and dynamic characteristics of the Web. Estimation errors within a reasonable range are usually acceptable. We employ effective sampling methods to estimate the parameters.

4.1.1. Estimating query cost

To estimate the local cost QC_{L} of a query, we employ a query sampling method similar to the one presented in references [Zhu and Larson 1994, 1996, 1998]. The idea is to divide queries into classes according to their access methods. Since the costs of queries in the same class follow a similar performance pattern, they can be estimated by the same cost formula. To derive a cost formula for each query class, a set of sample queries are drawn from the class and their costs are used in regression analysis. The query sampling method in references [Zhu and Larson 1994, 1996, 1998] was developed for a multidatabase environment where the access method used for a query is unknown due to local autonomy. Fortunately, the access methods for Web queries to be executed in the Query Processor here are known. A better classification is therefore expected, which will in turn yield better cost formulas. For example, we can have the following two classes of queries described in reference [Chen *et al.* 1998]:

Q_1: Queries requiring one pass scanning, e.g., "*SELECT NODE URL WHERE 'Computer', 'Database' IN Content AND Type = TEXT/HTML*". This query is to find the URLs of HTML documents that contain words '*Computer*' and '*Database*'. The query access method used for the query scans the full text index to match keywords '*Computer*' and '*Database*' and extracts the corresponding URLs of HTML documents.

Q_2: Queries requiring multiple-pass scanning, e.g., "*SELECT NODE URL WHERE 'Object Oriented' IN Content AND Type = TEXT/HTML AND X IN Link AND 'OODB' in X.Content*". This query is to find URLs of HTML documents which contain word '*Object Oriented*' and links to other documents that contain '*OODB*'. The access method used for the query scans an index for marching documents that contains '*Object Oriented*', and for each of them, checks whether any of their linked documents contains '*OODB*' by using an inverted index.

It is clear that the costs of queries in the first class follow a different performance pattern from the costs of queries in the second class. Hence their cost formulas are different. As suggested in references [Zhu and Larson 1994, 1996, 1998], to obtain more accurate estimates, each query class can be further divided into smaller classes when more information is available.

The remote cost[3] QC_{N} of a query is determined by network delay and the number and sizes of Web documents to be fetched. A simple estimate is $QC_{\mathrm{N}} = T_{\mathrm{nd}} \cdot (\sum_{i=1}^{M} |D_i|)$, where $|D_i|$ is the size of Web document D_i $(i = 1, 2, \ldots, M)$ to be fetched for the query. The estimation of network delay T_{nd} is to be discussed below. The sampling results in references [Bray 1996; Woodruff *et al.* 1996] show that the sizes of most Web documents are about 2 K, and the average size is 6–7 K. Based on such statistics, we employ the following simple rule to estimate the size of Web document D_i: if $|D_i|$ is available in the MetaDB, use it; otherwise let $|D_i| = 2$ K. After $|D_i|$ has been fetched, the actual size $|D_i|$ will be kept in the MetaDB. Cost estimation can also be done by using the method suggested in reference [Mendelzon *et al.* 1996], which is based on the concept of "query locality".

4.1.2. Estimating network delay

Network delay is determined by network configurations, load, and packet length. In Internet, network delay oscil-

[3] Note that $QC_{\mathrm{N}} = 0$ if the query searches the local database only.

lates dramatically due to the frequent changes of network configurations and load. In fact, estimating network delay from the local server to all nodes in Internet is unrealistic since there are almost infinite number of nodes in Internet with a 32-bit IP address space. We notice that there are two properties of network load. The first one is that network load is relatively stable within a certain period of time in a day although it may change dramatically from one period to another. For example, network load is high during working hours in a day and becomes low during the sleeping time at night. Another property of network load is that the load within the same autonomous system or subnetwork has some similarities. Based on these properties, we estimate T_{nd} values for different autonomous systems and subnetworks according to their numbers in IP addresses by transmitting a large number of testing packets, with size ranging from several bytes to hundreds of kilobytes, in different period between the local server and other interesting nodes in the Internet. The estimated values of T_{nd} will be kept in the MetaDB.

4.1.3. Estimating size of query queue

To estimate the size of a query queue, the system needs to know the arriving pattern of queries, including the number of arriving queries in a time interval and the distribution of arriving queries from different query classes.

Let $QN_{t_0}(t)$ denote the number of arriving queries in interval $[t_0, t)$ and $\|QL_{\text{w}}\|$ denote the size of query queue QL_{w}. The arriving of queries can be considered as a random Poisson process. Using a statistical method, we can obtain the average number of arriving queries λ in a time unit, so-called the strength of arriving stream of queries. Then the probability model of k arriving queries from t_0 to t is

$$P_k(t_0, t) = \frac{\lambda(t-t_0)^k}{k!} \mathrm{e}^{-\lambda(t-t_0)}, \quad t > t_0,\ k = 0, 1, 2, \ldots.$$

From the arriving pattern we can determine $\|QL_{\text{w}}\|$ according to an acceptable value for k that minimizes the number of queries which cannot enter the queue due to the low capacity of the queue.

For simplicity, we assume that arriving queries from different classes follow the uniform distribution. In practice, however, more queries may come from query classes $C^{(2)}$ and $C^{(3)}$ than from $C^{(1)}$ or $C^{(4)}$.

The size $\|QL_{\text{a}}\|$ of query queue QL_{a} is limited by system resources. Let $\theta = \|QL_{\text{a}}\|/\|QL_{\text{w}}\|$, which can be adjusted according to the system load.

4.2. Quality control protocol

As discussed in section 2.2, the distance function Δ for parameter vectors is used to determine the quality of query processing. To achieve a better quality, the quality controller needs to guide the query processor to process queries according to the current values of quality parameters.

Let U be the set of all quality parameters and $P_t(Q)$ be the value of quality parameter P for query Q at time t. U can be divided into the following three classes based on the behaviors of quality parameters during query processing:

- **Class I**: $\Gamma_{\text{I}} = \{SZ, NS, ND\}$. The main characteristic of quality parameters in this class is that the behavior of such a quality parameter for any query never becomes worse as query processing proceeds, that is, $\delta(P_t(Q), P_{t-1}(Q)) \geqslant 0$ for $P \in \Gamma_I$ and any query Q.
- **Class II**: $\Gamma_{\text{II}} = \{RT, TT, FR, GR\}$. The main characteristic of quality parameters in this class is that the behavior of such a quality parameter for any query never becomes better as query processing proceeds, that is, $\delta(P_t(Q), P_{t-1}(Q)) \leqslant 0$ for $P \in \Gamma_{\text{II}}$ and any query Q. Note that the current value of FR is the duration of existence for the oldest item accessed so far, which cannot exceed a user-specified value. Similarly, the current value of GR is the minimum set of HTML tags searched so far, which should contain at least the user-specified set (value).
- **Class III**: $\Gamma_{\text{III}} = \{AC, RL, RR\}$. The main characteristic of quality parameters in this class is that the behavior of such a quality parameter for a query is uncertain during query processing, i.e., it sometimes becomes better and sometimes becomes worse. In other words, it is possible that $\delta(P_t(Q), P_{t-1}(Q)) \cdot \delta(P_{t+1}(Q), P_t(Q)) < 0$ for $P \in \Gamma_{\text{III}}$ and a query Q at a time t.

For a given query Q and a quality parameter P, a local processing strategy $S_{\text{c}}^{\text{L}}(P, Q)$ is determined by the current value of P as follows:

- $S_{\text{c}}^{\text{L}}(P, Q) = \text{GO\_ON}$, i.e., continue to process, if $P \in \Gamma_{\text{I}}$ and $P_{\text{c}}(Q)$ is worse than the user-specified value $P_{\text{r}}(Q)$ (i.e., $\delta(P_{\text{c}}(Q), P_{\text{r}}(Q)) < 0$). Since further processing usually makes the quality of parameter P better (in fact, never makes it worse), query processing should continue.
- $S_{\text{c}}^{\text{L}}(P, Q) = \text{ABORT}$, i.e., abort the processing, if $P \in \Gamma_{\text{II}}$ and $P_{\text{c}}(Q)$ is worse than the user-specified value $P_{\text{r}}(Q)$ (i.e., $\delta(P_{\text{c}}(Q), P_{\text{r}}(Q)) < 0$). Since the current value (quality) of P is worse than the user-specified value and further processing never improves the quality of P, query processing should be aborted.
- $S_{\text{c}}^{\text{L}}(P, Q) = \text{WATCH}$, i.e., inspect the current behavior of P and take an action accordingly, if $P \in \Gamma_{\text{III}}$ and $P_{\text{c}}(Q)$ is worse than the user-specified value $P_{\text{r}}(Q)$ (i.e., $\delta(P_{\text{c}}(Q), P_{\text{r}}(Q)) < 0$). Since the behavior of P is uncertain, inspection of its value is needed. Assume that query Q has been executed on the CPU for k time slices so far. Let t_i, t_i' $(i = 1, \ldots, k)$ be the starting time and ending time of the ith time slice. Let

$$POS_{\text{c}}(Q) = \begin{cases} \sum_{\delta(P_{t_i'}(Q), P_{t_i}(Q)) \geqslant 0,\ 1 \leqslant i \leqslant k} \delta\big(P_{t_i'}(Q), P_{t_i}(Q)\big), \\ \qquad \text{if there exists } \delta\big(P_{t_i'}(Q), P_{t_i}(Q)\big) \geqslant 0, \\ 0, \quad \text{otherwise,} \end{cases}$$

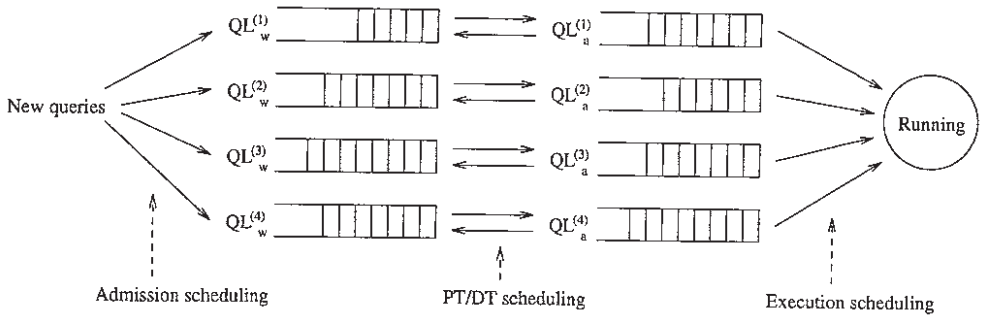

Figure 3. Query queue management in Quality Controller.

and

$$NEG_c(Q) = \begin{cases} \sum_{\delta(P_{t_i'}(Q),P_{t_i}(Q))<0,\ 1\leqslant i\leqslant k} \delta\big(P_{t_i}(Q), P_{t_i'}(Q)\big), \\ \quad \text{if there exists } \delta\big(P_{t_i'}(Q), P_{t_i}(Q)\big) < 0, \\ 0, \quad \text{otherwise.} \end{cases}$$

$POS_c(Q) < NEG_c(Q)$ implies that the overall negative achievement is larger than the overall positive achievement. In such a case, query processing should be aborted. Otherwise, query processing may continue. This policy guarantees that the overall behavior of parameter P is positive, although it may sometimes become worse and sometimes become better.

- $S_c^L(P, Q) =$ FINISH, i.e., terminate the processing normally, if $P_c(Q)$ is better than $P_r(Q)$ (i.e., $\delta(P_f(Q), P_r(Q) > 0)$. Since the user-specified requirement for P has been met, query processing can be finished.

The local processing strategy $S_c^L(P, Q)$ for query Q is, in fact, only a vote from the individual quality parameter P. Local processing strategies for query Q from different quality parameters may not be consistent. They have to be combined to yield a consistent global processing strategy for the query. The following quality control protocol is adopted to determine the global processing strategy $S_c^G(Q)$ for query Q based on the current values of a set of quality parameters $P_1, P_2, \ldots, P_n$.

Quality control protocol

1. $S_c^G(Q) =$ GO_ON if and only if (a) there exists one j such that $S_c^L(P_j, Q) =$ GO_ON and (b) $S_c^L(P_i, Q) =$ GO_ON or FINISH for every $i \in \{1, \ldots, j-1, j+1, \ldots, n\}$.
2. $S_c^G(Q) =$ ABORT if and only if there exists a j such that $S_c^L(P_j, Q) =$ ABORT.
3. $S_c^G(Q) =$ WATCH if and only if (a) there exists one j such that $S_c^L(P_j, Q) =$ WATCH and (b) there does not exist any i in $\{1, 2, \ldots, n\}$ such that $S_c^L(P_j, Q) =$ ABORT.
4. $S_c^G(Q) =$ FINISH if and only if $S_c^L(P_i, Q) =$ FINISH for every $i \in \{1, 2, \ldots, n\}$.

The global strategy $S_c^G(Q)$ determined by this protocol will be employed in the execution scheduling algorithm in section 4.4.3.

4.3. Managing queues

A multi-queue technique is designed to meet the challenges for scheduling queries with quality parameters. For each query class $C^{(i)}$ $(i = 1, 2, 3, 4)$ in section 2.3, there are one waiting queue $QL_w^{(i)}$ and one active queue $QL_a^{(i)}$. The sizes $\|QL_w^{(i)}\|$ of the waiting queues $QL_w^{(i)}$ are determined in the way described in the last subsection. The sizes $\|QL_a^{(i)}\|$ of the active queues $QL_a^{(i)}$ are then calculated as $\|QL_a^{(i)}\| = \|QL_w^{(i)}\| \cdot \theta$. Figure 3 illustrates the management of queues in the Quality Controller.

A new query is admitted to the waiting queue $QL_w^{(i)}$ for that query type based on the first-come-first-served policy. Queries in $QL_w^{(i)}$ are promoted to $QL_a^{(i)}$, and queries in $QL_a^{(i)}$ may be demoted to $QL_w^{(i)}$. Promotion/demotion is based on priorities for queries. These priorities reflect the quality requirements for parameters *RT* and *TT*. Each queue $QL_a^{(i)}$ also has a scheduling priority with $QL_a^{(1)}$ having the highest priority. Scheduling for queries in $QL_a^{(i)}$ to run on the CPU is based on the scheduling priority of the queue as well as all quality parameter requirements. The multi-queue model guarantees both the privilege of a higher priority queue and a fair scheduling among queries in the queues.

Primitive operations on queues are listed as follows:

- ADD(Q, i) – append query Q to queue $QL_w^{(i)}$ (i.e., change its state to W).
- EXCHANGE(Q_1, Q_2) – exchange query Q_1 in a waiting queue $QL_w^{(i)}$ with a query Q_2 in the corresponding active queue $QL_a^{(i)}$ (i.e., exchange their states).
- FINISH(Q) – finish the processing of query Q (i.e., remove it from the relevant $QL_a^{(i)}$ and move it to state T) and return its result.
- GETCLASS(Q) – return the class of query Q. The valid return values are 1, 2, 3 and 4 which represent classes $C^{(1)}, C^{(2)}, C^{(3)}$ and $C^{(4)}$, respectively.
- REJECT(Q) – reject query Q in state WA to enter the relevant $QL_w^{(i)}$ (i.e., move it to state T) and notify its user.
- MOVE(Q, S_1, S_2) – move query Q from state S_1 (i.e., from its associated queue) to S_2 (i.e., to its associated queue), where $S_1, S_2 \in \{\text{WA}, \text{W}, \text{A}, \text{R}\}$.
- ABORT(Q) – abort query Q (i.e., remove it from the relevant queue and move it to state T) and return the

Algorithm 1. Admission control for query Q

```
1.  k = GETCLASS(Q);
2.  if N_w^(k) = ||QL_w^(k)|| then REJECT(Q);
3.  if TT_r(Q) ≠ NULL and TT_r(Q) < QC(Q) then REJECT(Q);
4.  if RT_r(Q) ≠ NULL and N_a^(k) = ||QL_a^(k)|| then {
5.    let S = {Q | Q is a query in QL_a^(k) and RT_f(Q) = NULL};
6.    if ||S|| = ||QL_a^(k)|| then {
7.       V = min_{Q'∈S}{(RT_r(Q') − ET(Q'))};
8.       if RT_r(Q) < V then REJECT(Q);
9.    }
10. }
11. if SZ_r(Q) > SZ_max or NS_r(Q) > NS_max or ND_r(Q) > ND_max then REJECT(Q);
12.   // SZ_max, NS_max and ND_max are the maximum values allowed in the system//
13. ADD(Q,k).
```

Figure 4. Admission scheduling.

partial result (if any) together with its reached values of quality parameters to the user.

- EXECUTE(Q) – execute query Q on the CPU for a time slice.

4.4. Scheduling queries

As mentioned before, there are three types of scheduling in the quality controller: admission scheduling, PT/DT scheduling, and execution scheduling. The goals of scheduling are (1) to increase the number of queries that meet their quality requirements; (2) to maximize the quality of a query processing if system resources permit; and (3) to improve the overall system performance whenever possible. The first goal is the most important one. To achieve this goal, a few queries may have to be aborted in order to make more queries meet their quality requirements.

In the following discussion, let $N_w^{(i)}$ and $N_a^{(i)}$ be the numbers of queries in queues $QL_w^{(i)}$ and $QL_a^{(i)}$ $(i = 1, 2, 3, 4)$, respectively. Clearly, $N_w^{(i)} \leqslant \|QL_w^{(i)}\|$ and $N_a^{(i)} \leqslant \|QL_a^{(i)}\|$. Let $Q_w^{(i,j)}$ $(j = 1, 2, \ldots, N_w^{(i)})$ and $Q_a^{(i,j)}$ $(j = 1, 2, \ldots, N_a^{(i)})$ be the jth query in queues $QL_w^{(i)}$ and $QL_a^{(i)}$, respectively. $ET(Q)$ is the elapse time since for query Q.

4.4.1. Admission scheduling

In admission scheduling, user queries are considered on the first-come-first-served base. A query Q is admitted to a waiting queue if the user-specified quality parameter requirements are acceptable within the limitations of system resources. The admission control algorithm is given in figure 4. Note that $RT_f(Q)$ = NULL if the user has not received any result (response) for query Q from the system yet.

Line 2 rejects the admission of a query when the relevant waiting queue is full. Line 3 rejects the admission of a query if it is anticipated that the user-specified turnaround time for the query will not be met. Lines 4–10 reject the admission of a query if the following three conditions are true: (1) the relevant active queue is full, which implies that the system is busy; (2) none of the relevant active queries has met its response time requirement, which suggests that the system should give a higher priority to active queries in the system rather than a new query; and (3) the required response time of the new query is shorter than the remaining time allowed for the required response time of any active query in the system, which implies that the new query, if admitted, may compete for the system resources with the active queries to affect their response time requirements to be met. Clearly, both line 3 and lines 4–10 incorporate the current system load into the admission control, which prevents a greedy user from specifying unrealistic *RT* and *TT* quality requirements without considering other users or system resources. Line 11 rejects the admission of a query if some user-specified quality parameter values exceed the relevant system-allowed maximum values. Note that a system usually does not have any restrictions on the values of quality parameters *GR*, *AC*, *RL*, *FR* and *RR*. Hence these parameters are not checked in the admission control.

Upon receiving a reject response, the user may want to relax his/her quality requirements in order for the query to be processed. This improves the situation in which the user waits a long time for an unsatisfactory result because of the violation of some quality requirements. Our system displays the acceptable ranges for the rejected quality parameters when a user query is rejected. The starvation problem is thus solved in this way.

4.4.2. Promotion/demotion scheduling

The promotion of a query $Q_w^{(i,j)}$ in queue $QL_w^{(i)}$ occurs in two cases: (1) the capacity of queue $QL_a^{(i)}$ is not full, i.e., $N_a^{(i)} < \|QL_a^{(i)}\|$, which is called the simple promotion; (2) the capacity of queue $QL_a^{(i)}$ is full, i.e., $N_a^{(i)} = \|QL_a^{(i)}\|$, but the quality controller decides to exchange $Q_w^{(i,j)}$ with a query $Q_a^{(i,k)}$ (demoted) in queue $QL_a^{(i)}$, which is called promotion/demotion exchanging.

Note that *RT* and *TT* are two time-sensitive quality parameters. If a query is not processed in time, its *RT* and *TT* quality requirements may fail. Therefore, queries in each queue $QL_w^{(i)}$ are sorted in the ascending order[4] of values for the pair $(RT - ET, TT - ET)$. In other words, a query Q with a smaller $(RT_r(Q) - ET(Q), TT_r(Q) - ET(Q))$ value is given a higher priority in each waiting queue. Note that if

[4] The normal ordering for multiple fields is employed, i.e., pairs are sorted according to the first field, then the second field.

Algorithm 2. Simple promotion

```
1.  for i = 1 to 4 do {
2.      if N_a^(i) < ||QL_a^(i)|| then {
3.          while ((N_w^(i) > 0) and
4.                  ((TT_f(Q_w^(i,1)) - ET(Q_w^(i,1))) < 0
5.                    or (RT_f(Q_w^(i,1)) = NULL and (RT_r(Q_w^(i,1)) - ET(Q_w^(i,1))) < 0 )))
6.          do ABORT(Q_w^(i,1));
7.             // Note: N_w^(i) and QL_a^(i) have automatically been adjusted //
8.          if N_w^(i) > 0 then MOVE(Q_w^(i,1), W, A)
9.      }
10. }.
```

Figure 5. Simple promotion scheduling.

$RT_r(Q) = \text{NULL}$ (or $TT_r(Q) = \text{NULL}$) for query Q, a large system default value is assigned to $RT_r(Q)$ (or $TT_r(Q)$). Such a query has the lowest priority initially, but its priority grows as time elapses.

The simple promotion algorithm is given in figure 5. This algorithm first removes those queries whose *TT* or *RT* quality requirement has been violated (lines 3–6). It then promotes a query from each waiting queue to its corresponding active queue based on priorities (line 8). Note that the values of quality parameters other than *RT* and *TT* never change when the queries are in the waiting state.

It is possible that a query Q_1 in a waiting queue has a higher *RT*/*TT* quality requirement than that of a query Q_2 in the corresponding active queue. To avoid "thrashing", the quality controller will still keep Q_2 in the active state unless the difference between two *RT*/*TT* quality requirements is very big (e.g., exceed a threshold). In the latter case, Q_1 will be promoted to the active state and Q_2 will be demoted to the waiting state.

The following definition gives the conditions when a query is considered to be more urgent than another in terms of response time.

Definition 3. Let Q_i $(i = 1, 2)$ be two queries whose *RT* quality requirements are not violated; that is, either $RT_r(Q_i) > ET(Q_i)$ if $RT_f(Q_i) = \text{NULL}$, or $RT_r(Q_i) \leqslant RT_f(Q_i)$. Q_1 is said to be more urgent than Q_2 in terms of response time with respect to a threshold M (> 0) if one of the following two conditions holds:

(1) $RT_f(Q_1) = \text{NULL}$ and $RT_f(Q_2) \neq \text{NULL}$;

(2) $RT_f(Q_1) = RT_f(Q_2) = \text{NULL}$ and

$$\big(RT_r(Q_2) - ET(Q_2)\big) - \big(RT_r(Q_1) - ET(Q_1)\big) > M.$$

Condition (1) implies that Q_2 has already met its *RT* quality requirement (the system tries to improve other quality parameters), while Q_1 has not met its *RT* quality requirement. Condition (2) says that neither query has met its *RT* quality requirement, but Q_1 has less remaining time (with respect to threshold M) allowed for *RT* than Q_2 does. Note that query Q_i would not be considered to be urgent in terms of response time if $RT_f(Q_i) \neq \text{NULL}$ because this implies that the *RT* quality requirement of Q_i has been met already.

The following definition gives the condition when a query is considered to be more urgent than another in terms of turnaround time.

Definition 4. Let Q_i $(i = 1, 2)$ be two queries whose *TT* quality requirements are not violated; that is, $TT_r(Q_i) > ET(Q_i)$. Q_1 is said to be more urgent than Q_2 in terms of turnaround time with respect to a threshold M (> 0) if

$$\big(TT_r(Q_2) - ET(Q_2)\big) - \big(TT_r(Q_1) - ET(Q_1)\big) > M.$$

The condition essentially says that Q_1 has much less remaining time allowed for *TT* than Q_2 does. Note that $TT_f(Q_i) \neq \text{NULL}$ is not considered in definition 4 because it implies Q_i is completed (i.e., no longer in the system).

The promotion/demotion exchanging algorithm is given in figure 6. Lines 3–5 check if the *TT* or *RT* quality requirement for the query being considered in $QL_w^{(i)}$ is violated. If so, the query is aborted. Lines 8–10 check if the *TT* or *RT* quality requirement for the query being considered in $QL_a^{(i)}$ is violated. If so, the query is aborted. Lines 12–17 exchange queries $Q_w^{(i,j)}$ and $Q_a^{(i,k)}$ if the former is more urgent than latter in terms of both response time and turnaround time, by definitions 3 and 4. Note that the system thresholds are set sufficiently large to avoid frequent exchanging.

4.4.3. Execution scheduling

Queries in the active queues take turns running on the CPU. When the time slice for a running query is over, execution scheduling needs to determine the next query to be executed on the CPU.

Our execution scheduling is based on the priorities of queries. A priority value is a nonnegative real number. The smaller the priority value is, the higher the priority. Several factors are considered in determining fair priorities for queries.

Query classes. As shown in table 2, there are different types of queries in the Web. Each query class has its own characteristics. One query class is usually considered to be more important than another in a Web query system, depending on the objective of the system. For example, the importance for query classes $C^{(1)}$, $C^{(2)}$, $C^{(3)}$ and $C^{(4)}$ decreases in our system. To incorporate such a preference

Algorithm 3. *PT/DT* **exchanging**

```
1.  for i = 1 to 4 do {
2.      for j = 1 to N_w^(i) do {
3.          if ( (TT_r(Q_w^(i,j)) − ET(Q_w^(i,j))) < 0
4.              or ( RT_f(Q_w^(i,j)) = NULL and (RT_r(Q_w^(i,j)) − ET(Q_w^(i,j))) < 0 ) )
5.          then ABORT(Q_w^(i,j))
6.          else {
7.              for k = 1 to N_a^(i) do {
8.                  if ( (TT_r(Q_a^(i,k)) − ET(Q_a^(i,k))) < 0
9.                      or ( RT_f(Q_a^(i,k)) = NULL and (RT_r(Q_a^(i,k)) − ET(Q_a^(i,k))) < 0 ) )
10.                 then ABORT(Q_a^(i,k))
11.                 else {
12.                     if (Q_w^(i,j) is more urgent than Q_a^(i,k) in terms of both RT
13.                     and TT with respect to some system-defined thresholds)
14.                     then {
15.                         EXCHANGE(Q_w^(i,j), Q_a^(i,k));
16.                         break;   // exit the current loop //
17.                     }
18.                 }
19.             }
20.         }
21.     }
22. }.
```

Figure 6. Promotion/demotion exchange scheduling.

into execution scheduling, a class priority value $\alpha^{(i)} > 0$ is assigned to each query class $C^{(i)}$ $(i = 1, 2, 3, 4)$ such that $\alpha^{(1)} < \alpha^{(2)} < \alpha^{(3)} < \alpha^{(4)}$.

Time needed to finish. For a query Q that has not violated any quality parameter requirement, the distance from the vector of current quality parameter values to the vector of required quality parameter values is

$$\Delta\big(\vec{P}_{\mathrm{r}}(Q), \vec{P}_{\mathrm{c}}(Q)\big) > 0.$$

Assume that the last execution of Q on the CPU starts at time t and ends at time t', i.e., $t' - t$ is a time slice. Let $\vec{P}_t(Q)$, $\vec{P}_{t'}(Q)$ be the vectors of quality parameter values at time t and t', respectively. Then

$$V_t^{t'}(Q) = \frac{\text{distance from } \vec{P}_t(Q) \text{ to } \vec{P}_{t'}(Q)}{t' - t} = \frac{\Delta(\vec{P}_{t'}(Q), \vec{P}_t(Q))}{t' - t}$$

is the running velocity of query Q from time t to t'. If $V_t^{t'}(Q) > 0$, the vector of quality parameter values of Q moves towards the vector of required quality parameter values. Otherwise, the vector moves towards the opposite direction. Note that $t' - t$ is a constant since the scheduling time slice is not changed in our system. If Q has never been executed, the system assign an initial velocity $V_0 > 0$ to Q. Without loss of generality, assume $V_t^{t'}(Q) \neq 0$.[5]

Using the distance and velocity, the time (i.e., the number of time slices) needed to finish the query (i.e., achieve the required quality parameter values) from now can be estimated as

$$\theta_{\mathrm{c}}(Q) = \frac{\text{distance}}{\text{velocity}} = \begin{cases} \Delta(\vec{P}_{\mathrm{r}}(Q), \vec{P}_{\mathrm{c}}(Q))/V_t^{t'}(Q), & \text{if } V_t^{t'}(Q) > 0, \\ 3\Delta(\vec{P}_{\mathrm{r}}(Q), \vec{P}_{\mathrm{c}}(Q))/|V_t^{t'}(Q)|, & \text{if } V_t^{t'}(Q) < 0. \end{cases}$$

This estimation assumes that if $V_t^{t'}(Q) > 0$, Q will run at the same velocity $V_t^{t'}(Q)$ for distance $\Delta(\vec{P}_{\mathrm{r}}(Q), \vec{P}_{\mathrm{c}}(Q))$ to reach $\vec{P}_{\mathrm{r}}(Q)$; if $V_t^{t'}(Q) < 0$, Q will run at the same velocity $|V_t^{t'}(Q)|$ for distance $\Delta(\vec{P}_{\mathrm{r}}(Q), \vec{P}_{\mathrm{c}}(Q))$ in the opposite direction, then come back to the current point, and then continue to go in the right direction to reach $\vec{P}_{\mathrm{r}}(Q)$, with total distance $3\Delta(\vec{P}_{\mathrm{r}}(Q), \vec{P}_{\mathrm{c}}(Q))$.

If we let priority values be proportional to time $\theta_{\mathrm{c}}(Q)$, queries with less time needed to finish are given higher priorities than queries with more time needed to finish. Hence the system throughput is improved. Since higher priorities correspond to smaller priority values (time) and "time = distance/velocity", queries with smaller distances and higher velocities are preferred during execution scheduling. Note that since vectors of quality parameter values are used in calculating $\theta_{\mathrm{c}}(Q)$, every quality parameter has influence on execution scheduling.

Remaining time for RT. Among all quality parameters, *RT* and *TT* are two parameters that are time-sensitive, that is, if a query is not processed in time, its *RT* and *TT* requirements will be violated. Hence queries with less remaining time to fulfill their *RT*/*TT* quality requirements should have higher priorities for execution scheduling. For a query Q, its current remaining time allowed for *RT* is defined by the following function:

$$\varphi_{\mathrm{c}}(Q) = \begin{cases} RT_{\mathrm{r}}(Q) - ET(Q), & \text{if } RT_{\mathrm{f}}(Q) = \text{NULL}, \\ ET(Q) - RT_{\mathrm{f}}(Q), & \text{otherwise.} \end{cases}$$

[5] If $V_t^{t'}(Q) = 0$, consider the last non-zero velocity.

Algorithm 4. Execution scheduling

```
1.  while ('true') do {
2.     for i = 1 to 4 do {
3.        for j = 1 to N_a^(i) do {
4.           case S_c^G(Q_a^(i,j)) of
5.              ABORT :  ABORT(Q_a^(i,j));
6.              FINISH :  if there are queries in QL_w^(i) then FINISH(Q_a^(i,j));
7.              WATCH :  if there exists P ∈ Γ_III
8.                          such that POS_c(P(Q_a^(i,j))) < NEG_c(P(Q_a^(i,j)))
9.                       then ABORT(Q_a^(i,j));
10.             default : adjust Φ_c(Q_a^(i,j))
11.          end case
12.       }
13.    }
14.    adjust QL_1;
15.    EXECUTE(Q_a^(i_1,j_1));
16. }.
```

Figure 7. Execution scheduling.

We let the priority value of Q be proportional to the value of $\varphi_c(Q)$. Since $\varphi_c(Q)$ decreases as $ET(Q)$ increases for $RT_f(Q) = \text{NULL}$, query Q will receive a higher and higher priority as time elapses before its RT requirement is met. Since $\varphi_c(Q)$ increases with $ET(Q)$ for $RT_f(Q) \neq \text{NULL}$, query Q will receive a lower and lower priority as time elapses after its RT requirement is met.

Remaining time for TT. Like RT, the remaining time allowed for TT should also be reflected in the priority of a query. For a query Q, its current remaining time allowed for TT is defined by the following function:

$$\psi_c(Q) = TT_r(Q) - ET(Q).$$

Note that $ET(Q) \leqslant TT_f(Q) \leqslant TT_r(Q)$ must be true for a query processing without violating the TT quality requirement. We let the priority value of Q be proportional to the value of $\psi_c(Q)$, that is, the priority of Q becomes higher and higher as time elapses.

Therefore, the current priority value of query $Q_a^{(i,j)}$ in queue $QL_a^{(i)}$ is defined as follows:

$$\Phi_c\big(Q_a^{(i,j)}\big) = \alpha_i + C_1 \cdot \theta_c\big(Q_a^{(i,j)}\big) + C_2 \cdot \varphi_c\big(Q_a^{(i,j)}\big) + C_3 \cdot \psi_c\big(Q_a^{(i,j)}\big),$$

where C_1, C_2 and C_3 are constants.

Using $\Phi_c(Q_a^{(i,j)})$, all queries $Q_a^{(i,j)}$ can be sorted to form a logical queue $QL_l = (Q_a^{(i_1,j_1)}, Q_a^{(i_2,j_2)}, \ldots, Q_a^{(i_k,j_k)})$, where $k = \sum_{i=1}^{4} N_a^{(i)}$ and $\Phi_c(Q_a^{(i_s,j_s)}) < \Phi_c(Q_a^{(i_{s+1},j_{s+1})})$. When the time slice for the running query is over, the quality controller updates $\Phi_c(Q_a^{(i,j)})$, adjusts QL_l, and executes $Q_a^{(i_1,j_1)}$.

The Quality Control Protocol discussed in section 4.2 determines a (global) processing strategy $S_c^G(Q)$ for query Q based on the current values of its quality parameters, which is employed in the scheduling algorithm given in figure 7.

Lines 2–13 scan all queries in the active queues to abort and finish some queries and adjust the priority values for remaining active queries. Line 14 adjusts logical queue QL_l using the new priority values. Line 15 executes the first query (with the highest priority) in QL_l. Note that when $S_c^G(Q_a^{(i,j)}) = \text{FINISH}$ in line 6, i.e., all quality requirements of $Q_a^{(i,j)}$ have been met (in other words, the 'correct' result has been obtained), the scheduler terminates $Q_a^{(i,j)}$ if the system load is high (i.e., $QL_w^{(i)}$ is not empty). However, the scheduler may continue to process such queries when system resources permit, which can further improve the quality of queries. When $S_c^G(Q_a^{(i,j)}) = \text{WATCH}$ in line 7, the scheduler aborts the query if the overall negative achievement is bigger than the overall positive achievement for a quality parameter in class III. (See section 4.2 for the definitions of $POS_c(\cdot)$ and $NEG_c(\cdot)$.)

Applying scheduling algorithms 1–4, the quality controller schedules queries for the query processor according to the quality requirements. The query processor then processes the queries and eventually returns the results to the users.

During query processing, some query optimization strategies such as buffer sharing, parallel processing, and execution plan dynamic improving can be applied. Although many conventional query optimization techniques can be utilized for processing Web queries, quality control raises some new challenges, especially the new optimization objective function is to maximize the distance function for a quality parameter vector instead of to minimize the response time only as in most conventional query optimizers. Due to the limitation of the paper length, query optimization for queries with quality requirements will be discussed in a separate paper.

5. Conclusion

Database-like queries in the Web allow users to find useful information from a huge number of Web data sources more effectively, compared with the conventional navigation approach. However, the quality of query processing in the Web is usually very low. For instance, a user often suffers from problems like unexpected long response time, unrelevant query results, and out-of-date information.

To tackle the challenge of low-quality Web query processing, we proposed a novel quality-controlled query processing method for Web queries. The idea is to allow a user to specify quality parameters when he/she submits a query. The Web query processing system first checks if user-specified quality parameter requirements are feasible. If not, the system informs the user at the beginning and recommends acceptable ranges for the rejected quality parameters. The system then schedules and processes the queries with quality parameters in such a way that user-specified quality parameter requirements are met. Since users can control the quality of their queries including response time and accuracy of the result, satisfactory query processing and results will be achieved, which implies that quality query processing is provided. Since Web resources change frequently, unexpected situations may occur during query processing. Hence, although most queries can meet their quality requirements, a few queries may have to be aborted during query processing due to system contention or other reasons. In such a case, the system will inform the relevant users as soon as possible and return partial query results together with their reached values of quality parameters. In any case, users will know the quality of query results that they have received.

The main contributions of this paper are summarized as follows:

- A framework of query processing with quality control is introduced. The relevant concepts are formalized.
- A set of quality parameters that can be specified with a Web query are identified.
- The distance functions to evaluate the goodness of quality parameters individually or collectively are defined.
- A quality control protocol for quality parameters in query processing is suggested.
- A set of quality controlled query scheduling algorithms including admission scheduling, promotion/demotion scheduling and execution scheduling are proposed.

A Web query processing prototype incorporating the quality control techniques discussed in this paper is under development. We believe that the proposed quality-controlled query processing method is a promising way to solve the uncertain and low-quality query processing problem in the Web.

The work reported here is only the beginning of more research that needs to be done in order to completely solve the problem of query processing with quality control in the Web. In the future, in addition to further improving the quality control algorithms, we also plan to study the issues on query processing with quality control on multimedia Web data and in a collaborative resource discovery system in which multiple Web query processing systems are involved in processing a Web query [Chen *et al.* 1998].

References

Ashish, N. and C. Knoblock (1997), "Wrapper Generation for Semi-Structured Internet Sources," In *The PODS/SIGMOD Workshop on Management of Semistructured Data*, Tucson, AZ.

Berners-Lee, T. (1996), "WWW: Past, Present, and Future," *IEEE Computer 29*, 10, 69–77.

Berners-Lee, T. and D. Conolly (1995), "Hypertext Markup Language – 2.0," IETF RFC 1866.

Bray, T. (1996), "Measuring the Web," *Computer Networks and ISDN Systems 28*, 7/11, 993–1005.

Chen, Y., H. Xu, and N. Wang (1998), "WWWDS: Towards Globalization of Distributed Data Sources over WWW," *Chinese Journal of Software 9*, 8, 566–573.

Chen, Y., B. Xu, and N. Wang (1998), "WebCORD: A Collaborative Resource Discovery System Model in Web," *Chinese Journal of Computer 21*, 4, 381–384.

Fiebig, T., J. Weiss, and G. Moerkotte (1997), "RAW: A Relational Algebra for the Web," In *The PODS/SIGMOD Workshop on Management of Semistructured Data*, Tucson, AZ.

Gosinski, T. and S. Avila (1997), "Implementing a Regional Traffic Data Management System," In *Proceedings of the Annual Conference of the Urban and Regional Information Systems Association*, Washington, DC, pp. 735–743.

Hammer, J., H. Garica-Molina, J. Cho, R. Aranha, and A. Crespo (1997), "Extracting Semistructured Information from the Web," In *The PODS/SIGMOD Workshop on Management of Semistructured Data*, Tucson, AZ.

Kim, K., J. Kim, J. Choi, and M. Sung (1995), "Application of GIS to Water Quality Management," In *GIS/LIS '95 Annual Conference and Exposition*, Vol. 2, Nashville, TN, pp. 554–562.

Konopnicki, D. and O. Shmueli (1995), "W3QS: A Query System for the World-Wide Web," In *Proceedings of the 21st International Conference on Very Large DataBases*, Morgan Kaufmann, Zurich, Switzerland, pp. 54–65.

Lamm, S.E., D.A. Reed, and W.H. Scullin (1996), "Real-Time Geographic Visualization of World Wide Web Traffic," *Computer Networks and ISDN Systems 28*, 7/11, 1457–1468.

Levy, A.Y., A. Rajaraman, and J.J. Ordille (1996), "Querying Heterogeneous Information Sources Using Source Descriptions," In *Proceedings of the 22nd International Conference on Very Large DataBases*, Morgan Kaufmann, Mumbai, IN, pp. 251–262.

Mendelzon, A.O., G.A. Mihaila, and T. Milo (1996), "Querying the World Wide Web," In *Proceedings of the 4th IEEE International Conference on Parallel and Distributed Information Systems*, Miami Beach, FL, pp. 80–91.

Vrbsky, S.V. (1994), "A Data Model for Approximate Query Processing of Real-Time Database," *Data & Knowledge Engineering 21*, 1, 79–102.

Winder, D. (1997), "Internet Search Engines," *PC Pro 34*, 212–219.

Woodruff, A., P.M. Aoki, E. Brewer, P. Gauthier, and L.A. Wowe (1996), "An Investigation of Documents from the World Wide Web," *Computer Networks and ISDN Systems 28*, 7/11, 963–980.

Yuwono, B. and D.L. Lee (1996), "Searching and Ranking Algorithms for Locating Resources on the World Wide Web," In *Proceedings of the 12nd IEEE International Conference on Data Engineering*, New Orleans, LA, pp. 164–171.

Zhu, Q. and P.-Å. Larson (1994), "A Query Sampling Method for Estimating Local Cost Parameters in a Multidatabase System," In *Proceedings of the 10th IEEE International Conference on Data Engineering*, Houston, TX, pp. 144–153.

Zhu, Q. and P.-Å. Larson (1996), "Building Regression Cost Models for Multidatabase Systems," In *Proceedings of the 4th IEEE International Conference on Parallel and Distributed Information Systems*, Miami Beach, FL, pp. 220–231.

Zhu, Q. and P.-Å. Larson (1998), "Solving Local Cost Estimation Problem for Global Query Optimization in Multidatabase Systems," *Distributed and Parallel Databases 6*, 4, 373–420.