Optimizing source-call ordering in Information Gathering Plans

Subbarao Kambhampati*& Senthil Gnanaprakasam
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287
email: {*rao,gsenthil*}@*asu.edu*
http://rakaposhi.eas.asu.edu/iii99.ps

Abstract

In this paper we consider the problem of optimizing the order in which source relations are joined in information gathering plans. This problem differs significantly from the traditional database query optimization problem, as sources on the Internet have a variety of access limitations and the execution cost in information gathering is affected both by network traffic and by the connection setup costs. We describe a way of representing the access capabilities of sources, and provide a greedy algorithm for ordering source calls that respects source limitations. Our algorithm also takes both access costs and traffic costs into account, without requring full source statistics. This algorithm is being evaluated in the context of *Emerac*, our prototype information gathering system.

Introduction

One way of modeling the task of gathering information on the Internet involves building a virtual global schema for the information that the user is interested in, and describing the accessible information sources as materialized views on the global schema.[1] The user query is posed in terms of the relations of the global schema. Since the global schema is virtual (in that its extensions are not stored explicitly anywhere), the gatherer needs to rewrite (or "fold" [Qia96]) the query such that all the EDB predicates in the rewrite correspond to the materialized view predicates that represent information sources (see Figure 1). Several researchers [LRO96, Qia96, KW96] have addressed this rewriting problem. Recent research by Duschka and his coworkers [DG97, DL97] provides a clean methodology for constructing information gathering plans for user queries posed in terms of a global schema. The plans produced by this methodology are "maximally contained" in that any other plan for answering the given query is contained in them.[2]

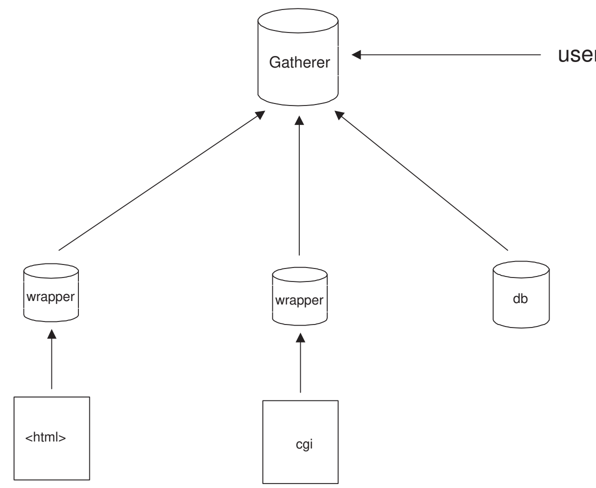

Figure 1: The information gatherer acts as an intermediary between the user and information sources on the Internet.

Generating source complete plans however is only a first step towards efficient information gathering. A crucial next step is that of query plan optimization. Given the autonomous and decentralized nature of the Internet, sources tend to have significantly overlapping contents (e.g. mirror sources), as well as varying access costs (premium vs. non-premium sources, high-traffic vs. low-traffic sources). Naive execution of maximally contained plans can be prohibitively costly, in terms of the network traffic, response time, and access costs (in the case of "premium" sources that charge for access).

Query optimization for information gathering on internet is the main focus of our *Emerac* project [LK99, LK98a]. Figure 2 provides a schematic illustration of the query planning and optimization process in *Emerac*. We start by generating a query plan using the source inversion techniques described Duschka [DG97, DL97]. This polynomial time process gives us a "maximally contained" query plan which serves as the input for the optimization methods. Our optimization step involves two phases: logical optimization and execution optimization. The logical optimization step

*This research is supported in part by NSF young investigator award (NYI) IRI-9457634, ARPA/Rome Laboratory planning initiative grant F30602-95-C-0247, Army AASERT grant DAAH04-96-1-0247, AFOSR grant F20602-98-1-0182 and NSF grant IRI-9801676. We thank Selçuk Candan for many helpful comments. An up-to-date version of this paper can be found at the URL listed in the title.

[1] See [LK97] for a tutorial.

[2] In other words, executing a maximally contained plan guarantees the return of every tuple satisfying the query that can be returned by executing any other query plan.

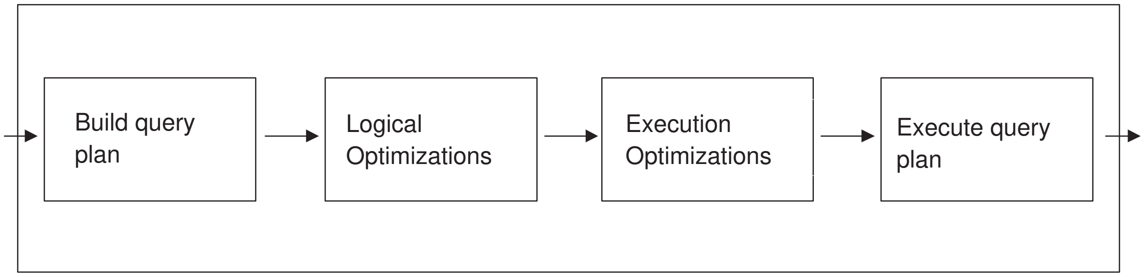

Figure 2: The full process of query planning

involves minimizing the maximally contained query plan such that access to redundant sources is removed. The *Emerac* logical optimization phase is discussed in detail in [LK99, LK98a]. In this phase, the recursive plans generated by Duschka's algorithm are greedily minimized so as to remove access to costly and redundant information sources, without affecting the completeness of the plan. The output of this phase is a minimized datalog program, whose EDB predicates correspond to source calls.

The second phase of *Emerac* optimization, which involves ordering the source calls in each of the datalog rules in the minimized plan, is the focus of the current paper. A crucial practical choice we have to make during the evaluation of datalog programs is the order in which predicates are evaluated. Ultimately plan execution in our context largely boils down to doing joins between the sources efficiently. Our objective is to reduce the cost of computing joins, where the cost is a function of the access cost (including connection time), traffic costs (the number of tuples transferred), and processing cost (the time involved in processing the data). Traditional join-ordering algorithms [Cha98, ÖV98] are not particularly relevant for *Emerac* for a variety of reasons:

- Join ordering algorithms assume that all sources are relational databases. The sources on the Internet are rarely fully relational and tend to support limited types of queries. These limitations need to be represented and respected by the join ordering algorithm.
- A corollary of source limitations is that it eliminates the possibility of shipping a relation to a source for joining. The processing of joins thus needs to be done locally.
- Join ordering algorithms in distributed databases typically assume that the cost of query execution is dominated by the number of tuples transferred during execution. Thus, the so-called "bound-is-easier" assumption makes good sense. In the Internet information gathering scenario, the cost of accessing sources tends to equal or dominate the execution cost. Thus we cannot rely solely on the bound-is-easier assumption and would need to consider the number of source calls.
- Typically, join ordering algorithms use statistics about the sizes of the various relations, as well as join selectivities, to compute an optimal order of joining. These techniques are not applicable in *Emerac*, as we typically do not have complete statistics about Internet information sources.
- The fact that source latencies make up a significant portion of the cost of execution argues for parallel (or "bushy") join trees instead of the "left-linear" join trees considered by the conventional algorithms [Cha98].

In the rest of the paper, we describe the *Emerac* approach to source-call ordering. We first describe a way of representing the access capabilities of sources. Next, we provide a greedy algorithm for ordering source calls that respects source limitations, and takes both access costs and traffic costs into account, without requring full source statistics. We then discuss related work and conclude.

Representing source access capabilities

As we mentioned, in the information gathering domain, the assumption that information sources are fully relational databases is rarely valid. An information source may now be a wrapped web page, a form interfaced database, or a fully relational database. A wrapped web page is a WWW document interfaced through a wrapper program to make it appear as a relational database. The wrapper retrieves the web page, extracts the relational information from it, then answers relational queries. Normal selection queries are not supported. A form-interfaced database refers to a database with an HTML form interface on the web which only answers selection queries over a subset of the attributes in the database. A WWW airline database that accepts two cities and two dates and returns flight listings is an example of a form interfaced database.

In *Emerac*, we use a simple way to inform the gatherer as to what types of queries on an information source would accept. We use the "\$" annotation to identify variables that must be bound, and "%" annotation to identify unselectable attributes (i.e., those that must not be bound). Thus a fully relational source would be adorned $source(X, Y)$, a form interfaced web-page that only accepts bindings for its first argument would be adorned $source(X, \%Y)$, while a wrapped web-page source would have all its attributes marked unselectable, represented as $source(\%X, \%Y)$. Finally, a form interfaced web-page that requires bindings for its first argument, and is able to do selections only on the second argument would be adorned as $source(\$X, Y, \%Z)$.

The "\$" and "%" annotations are used to identify feasible binding patterns for queries on a source, to establish generality relations between two binding patterns, and to ensure that soundness is preserved in pushing variable selection constraints (such as "$Y = 7$") into source calls. Given a source with annotations $S_1(\$X, \%Y, Z)$, only the binding patterns of the form S_1^{b--} are feasible (where "$-$" stands for either *b*ound or *f*ree argument). Similarly, we are not allowed to push selection constraints on Y to the source S1 (they must be filtered locally). Thus the call S_1^{bbf} is modeled as S_1^{bff} filtered locally with the binding on Y.

A binding pattern S^p is more general than S^q (written $S^p \succ_g S^q$, if every selectable (non "%"-annotated) variable that is free in q is also free in p, but not *vice versa*. Thus, for the source S_1 above, the binding pattern S_1^{bbf} is more general than the binding pattern S_1^{bfb} (while such a relation would not have held without "%" annotations [Ull89]). Intuitively, the more general a binding pattern, the higher the number of tuples returned by the source when called with that binding pattern. We ignore binding status of "%"-annotated variables since by definition they will not have any effect on the amount of data transferred. Finally, we define $\#(\alpha)$ as as the number of bound variables in α that are not %-annotated. Notice that "$\succ_g$" holds only between binding patterns of the same source while "$\#(.)$" can be used to relate binding patterns of different sources.

Issues in ordering source calls

We now consider the task of ordering the conjuncts in the individual rules of the minimized query plan (datalog program). We noted that the execution cost is a function of the access cost (including connection time), traffic costs (the number of tuples transferred), and processing cost (the time involved in processing the data). Thus, optimal plans will need to minimize:

$$\sum_s (C_a^s * n_s + C_t^s * D_s)$$

Where n_s is the number of times a source s has been accessed during the plan and C_a^s is the cost per access, and C_t^s is the per tuple transfer cost for source s, and D_s is the number of tuples transferred by s. We note that this cost metric imposes a tension between the desire to reduce network traffic, and the desire to reduce access costs. To elaborate, reducing the network traffic involves accessing sources with less general binding patterns. This in turn typically increases the number of separate calls made to a source, and leads to increased access cost. To illustrate this further, consider the subgoals:

$$\text{SOURCEA}(W, X) \wedge \text{SOURCEB}(X, Y)$$

Suppose that the query provides bindings for W. How should we access the sources? The conventional wisdom says that we should access SOURCEA first since it has more bound arguments. As a result of this access, we will have bindings for X which can then be fed into calls to SOURCEB. The motivation here is to reduce the costs due to network traffic. However, calling SOURCEA and using its outputs to bind the arguments of SOURCEB may also lead to a potentially large number of separate calls to SOURCEB[3] (one per each of the distinct X values returned by SOURCEA), and this can lead to a significant connection setup costs, thus worsening the overall cost. We need to thus consider both the access costs and the traffic costs in ordering the sources.

A greedy algorithm for ordering source calls

We made two important assumptions in designing the *Emerac* source-call ordering algorithm:

- Exact optimization of the execution cost requires access to source selectivity statistics. We assume that such statistics are not easily available.
- We assume that by default source access cost (rather than network traffic) is the dominating cost of the plan. This is a reasonable assumption given the large connection setup delays involved in accessing sources on the Internet. While the traffic costs can also be significant, this is offset to some extent by the fact that many data sources on the Internet do tend to have smaller extractable tables.[4]

If our assumption about the secondary importance of network traffic costs were always true, then we can issue calls to any source as soon as its binding constraints are met (i.e., all the variables requiring bindings have bindings available). Furthermore, we need only access the source with the most general feasible binding pattern (since this will reduce the number of accesses to the source). We provide an escape clause for this assumption (see below), as sometimes sources can transfer arbitrarily large amounts of data for calls with sufficiently general binding patterns.

High-traffic Binding Patterns: To ensure that we don't get penalized excessively for concentrating primarily on access costs, we also maintain a table HTBP of least general (w.r.t. "$\succ_g$") source binding patterns that are still known to be high-traffic producing. The general idea is to postpone calling a source as long as all the feasible binding patterns for that source supported by the currently bound variables are equal to or more general than a binding pattern listed in HTBP.

An underlying assumption of this approach is that while full source statistics are rarely available, one can easily gain partial information on the types of binding patterns that cause excessive traffic. For example, given a source that exports the relation

$$Book(Author, Title, ISBN, Subject, Price, Pages)$$

we might easily find out that calls that do not bind at least one of the first four attributes tend to generate high traffic.

[3] Unless SOURCEB accepts a list of possible values for X in a single call [GMLY99]

[4] Even those that have large tables regulate their data output, by paginating the tuples and sending them in small quanta (e.g., first 10 tuples satisfying the query), which will avoid the network congestion if the users needed only a certain percentage of the tuples satisfying the query.

```
Inputs: FBP: table of forbidden binding patterns
  HTBP: table of high traffic binding patterns
  V := all variables bound by the head;
  C[1···m]: Array where C[i] lists sources chosen at i^th stage;
  P[1···m]: Array where P[i] lists sources postponed at i^th stage
for i := 1 to m (where m is the number of subgoals) do begin
  C[i] := ∅; P[i] := ∅;
  for each unchosen subgoal S do begin
    B := All feasible binding patterns for S w.r.t. V and FBP
          sorted using "≻_g" relation.
    for each β ∈ B do begin
      if ∄_{β'∈HTBP} s.t.(β = β') ∨ (β ≻_g β')
      then begin
          Push S with binding pattern β into C[i];
          Mark S as "chosen";
          add to V all variables appearing in S;
      end
  end
  if B ≠ ∅ and S is not chosen
    then Push S^γ into P[i], where
        γ ∈ B has the maximum #(.) value;
  end
  if C[i] = ∅ and P[i] ≠ ∅
    then begin
      Take the source S^β ∈ P[i] with maximum #(.) value
      and push it into C[i];
      add to V all variables appearing in S;
    else fail
  end
  Return the array C[1..i].
```

Figure 3: A greedy source call ordering algorithm that considers both access costs and traffic costs.

A greedy algorithm to order source calls based on these ideas appears in Figure 3. It takes as input a rule with m subgoals and a given binding pattern for its head. The input also includes FBP, a table of forbidden binding patterns for each source (constructed from the "\$" annotations), and the table HTBP, which contains all source binding patterns that are known to be high-traffic producing. At each level i, the algorithm considers the feasible binding patterns of each unchosen source from most general to least general, until one is found that is not in HTBP. If such a binding pattern is found, that source, along with that binding pattern, is added to the set of selected sources at that level (maintained in the array of sets $C[i]$). If not, the source, along with the least general feasible binding pattern (given the "\$" restrictions as well as the currently avaialble bound variables), is temporarily postponed (by placing it in $P[i]$). If at the end of considering all unchosen sources at level i, we have not chosen at least one source (i.e., all of them have only high-traffic inducing binding patterns), then one of the sources from the list of postponed sources ($P[i]$) is chosen and placed in $C[i]$. This choice is made with the "bound-is-easier" assumption by selecting the source with the least general binding pattern (in terms of $\#(.)$).

Anytime a source is placed in $C[i]$, the set V of variables that currently have available bindings is updated. This ensures that at the next stage more sources will have feasible, as well as less general, binding patterns available. This allows the algorithm to progress since less general binding patterns are also less likely to be present in the HTBP table. Specifically, when $C[i]$ is empty and $P[i]$ is non-empty, we push only one source from $P[i]$ into $C[i]$ since it is hoped that the updated V will then support non-high-traffic binding patterns at the later stages. (By consulting HTBP, and the set of unchosen sources, the selection of the source from $P[i]$ can be done more intelligently to ensure that the likelihood of this occurrence is increased).

When the algorithm terminates successfully, the array C specifies which sources are to be called in each stage, and what binding patterns are to be used in those calls. Execution involves issuing calls to sources with the specified binding pattern; where each bound variable in the binding pattern is instantiated to all values of that variable collected upto that point during execution. If the bound variable is a %-annotated variable, then the call is issued without variable instantiation, and the filtering on the variable values is done locally. The tuples returned by the source calls in each stage are locally joined.

Notice that each element of C is a (possibly non-singleton) set of source calls with associated binding patterns (rather than a single source call)–thus supporting bushy joins. Thus, $C[i]$ may be non-empty for only a prefix of values from 1 to m. This parallelism cuts down the overall time wasted during connection delays. The complexity of our ordering algorithm is $O(n^2)$ where n is the length of the rule.

It is worth noting that the behavior of our algorithm depends on the contents of the HTBP table. When HTBP contains no binding patterns, the algorithm essentially concentrates on reducing the source accesses (similar to [YL99]). When all source binding patterns are listed in HTBP, the algorithm winds up focusing on the network traffic, and reduces to a variant of conjunct ordering by bound-is-easier assumption [Ull89].

Related Work

In our earlier work on query optimization for *Emerac*, we developed a method [LK98b] that assumes that the network traffic is the only dominant cost to be optimized, and thus uses an algorithm based on bound-is-easier assumption, modified suitably to take the selection restrictions into account. The algorithm in this paper is more general in that it allows for access costs and source latencies, which are an important aspect of sources on the Internet. In particular, the algorithm here reduces to a bound-is-easier type algorithm, if we assume that every source binding pattern is present in the HTBP table.

Our source-access ordering technique assumes that statistics regarding source relations are not easily available, and thus traditional join-ordering strategies are not applicable. An interesting alternative is to try and learn the source statistics through experience. Zhu and Larson [ZL96] describe techniques for developing regression cost models for multi-database systems by selective querying. Adali et. al [ACPS96] discuss how keeping track of rudimentary access statistics can help in doing cost-based optimizations.

Concurrent with our work, several research efforts [HKWY97, LYV+98, GMLY99, VP97, YL99] have considered the issue of optimizing queries in the presence of sources with limited query capabilities. In contrast to our approach of using "\$" and "%" annotations to represent query capabilities of sources, TSIMMIS system [LYV+98, YL99] lists all "legal" binding patterns entertained by a source, while GARLIC [HKWY97] system provides the query processing capabilities of sources in the form of a set of rules. More elaborate languages for representing source access capabilities are proposed in [GMLY99, VP97]. While some of these systems [YL99, GMLY99] provide more rigorous optimality guarantees than our greedy approach, none of these systems consider the source access costs and traffic costs together in doing optimization.

Conclusion

We presented a greedy algorithm for ordering source calls that respects source limitations, and takes both access costs and traffic costs into account. A novelty of our approach is the use of HTBP table, which can be seen as providing a way to maintain coarse information about the source relations, without requiring elaborate statistics. We are currently evaluating this technique within *Emerac* and hope to have empirical results by the time of workshop. We are also considering extensions to our approach so that it is able to effectively use any available source statistics. One idea is to generalize the HTBP table so that it gives weights to different binding patterns, where the weights are proportional to the actual traffic that will be produced.

References

[ACPS96] Sibel Adalı, Kasim S. Candan, Yannis Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proceedings of the ACM Sigmod International Conference on Management of Data*, pages 137–148, 1996.

[Cha98] Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 98*, pages 34–43, 1998.

[DG97] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '97*, pages 109 – 116, Tucson, AZ, May 1997.

[DL97] Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI*, Nagoya, Japan, August 1997.

[GMLY99] Hector Garcia-Molina, Wilburt Labio, and Ramana Yerneni. Capability sensitive query processing on internet sources. In *Proc. ICDE*, 1999.

[HKWY97] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *Proc. VLDB*, 1997.

[KW96] Chung T. Kwok and Daniel S. Weld. Planning to gather information. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996.

[LK97] Eric Lambrecht and Subbarao Kambhampati. Planning for information gathering: A tutorial survey. Technical Report ASU CSE TR 97-017, Arizona State University, 1997. rakaposhi.eas.asu.edu/ig-tr.ps.

[LK98a] Eric Lambrecht and Subbarao Kambhampati. Optimization strategies for information gathering plans. Technical Report ASU CSE TR 98-018, Arizona State University, 1998.

[LK98b] Eric Lambrecht and Subbarao Kambhampati. Optimizing information gathering plans. In *Proc. AAAI-98 Workshop on Intelligent Information Integration*, 1998.

[LK99] Eric Lambrecht and Subbarao Kambhampati. Optimizing recursive information gathering plans. In *Proc. IJCAI*, 1999.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases*, pages 251–262, Bombay, India, 1996.

[LYV+98] Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-Molina, Yannis Papakonstantinou, Jeffrey Ullman, and Murty Valiveti. Capability based mediation in tsimmis. In *Proc. SIGMOD*, 1998.

[ÖV98] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems (2nd ed)*. Prentice-Hall, 1998.

[Qia96] Xiaolei Qian. Query folding. In *Proceedings of the 12th International Conference on Data Engineering*, pages 48–55, New Orleans, LA, February 1996.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledgebase Systems*, volume 2. Computer Science Press, 1989.

[VP97] Vasilis Vassalos and Yannis Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In *Proc. VLDB*, 1997.

[YL99] Ramana Yerneni and Chen Li. Optimizing large join queries in mediation systems. In *Proc. International Conference on Database Theory*, 1999.

[ZL96] Qiang Zhu and Per-Ake Larson. Developing regression cost models for multidatabase systems. In *In Proceedings of PDIS*, 1996.