Query Optimization in Mobile Environments

Sumit Ganguly
Dept. of Computer Sciences
New Brunswick, NJ 08542

Rafael Alonso
MITL
Princeton, NJ 08540

e-mail: sumit@cs.rutgers.edu
Tel:(908) 932-4974
Fax: (908) 932-0537

December 15, 1993

Abstract

We consider the issue of optimizing queries for a distributed processing in mobile environment. An interesting characteristic of mobile machines is that they depend on battery as a source of energy which may not be substantial enough. Hence, the appropriate optimization criterion in a mobile environment considers both resource utilization and energy consumption at the mobile client.

In this scenario, the optimal plan for a query depends on the residual battery level of the mobile client and the load at the server. We approach this problem by compiling a query into a sequence of candidate plans, such that for any state of the client-server system, the optimal plan is one of the candidate plans.

A general solution is proposed by adapting the partial order dynamic programming search algorithm [17, 18] (p.o dp) such that the coverset of the query is the set of candidate plans. We propose two novel algorithms, namely, the *linear combinations* algorithm and the *linearset* algorithm (referred to as the *linear* algorithms) that compute the *linearset* of a query. The linearset of a query is an approximation to the coverset returned by p.o. dp.

We show, by means of simulation, that (1) the linearset is an excellent approximation of the coverset, (2) query compilation using the linear algorithms outperform query compilation using p.o. dp by factors ranging from 2 to 9, (3) the time taken to compile queries using the linear algorithms for the general optimization criterion is at most twice the time taken by a System R* like standard query optimizer search algorithm, and (4) the run time overhead incurred by the linear algorithms technique is minimal.

The techniques presented in the paper are of general applicability to multi-criterion optimization problems in distributed databases, where each criterion is an *additive metric*.

1 Introduction

Computing with mobile machines introduces novel technical challenges in the area of database systems [6] and resurrects some issues in distributed database systems. In this paper, we study the ramifications of one important characteristic of mobile computing, namely, the reliance of portable computers on batteries as the power source, on query processing and optimization. Normal use of such machines cause batteries to run out of power within several hours [32] and heavy use may substantially decrease this period. Moreover, significant advances in battery technology do not seem to be forthcoming.

An interesting feature of current portable computers is that the computer may run in at least 2 modes, such as full on (CPU running at 25 MHz) and doze (CPU running at 3 MHz), in the AMD Saturn computer [1]. Switching between CPU modes and/or suspending or powering the hard disk may be controlled by software from within an application program. We refer to the ratio of the power consumed in energy saving mode to the power consumed in normal mode as the *idling coefficient*. For current mobile machines, the idling coefficient lies between 0.1 to 0.6.

The emerging architecture of a mobile computing system is that of a distributed system in which a collection of mobile clients communicate with a stationary (and powered) server [6]. We consider a scenario in which the mobile client participates in distributed query processing with the server.

The appropriate optimization criterion in a mobile computing environment should take both energy consumed at the client and the resource utilization at the server into account. For example, a query may be compiled in at least two different ways, depending on the residual battery level of the client. The optimal plan for low battery level may place a greater burden of query processing on the server. We argue that the optimization criterion for a mobile computing environment is the following.

Maximize server throughput, such that client energy consumption $\leq$ client energy threshold, and server resource utilization $\leq$ server threshold.

The constraint on energy consumption arises as follows. Clients may not be concerned about energy when their *energy levels* are high, i.e., the remainder life of the battery is several hours etc. In this scenario, the energy threshold is high (or non-existent). If the energy level of the client is low, then the client's energy threshold is also low, i.e., the *energy threshold is a non-decreasing function of the energy level of the client.* The variable threshold on energy is one of the distinguishing elements of the mobile environment.

The constraint on server resource utilization is placed due to the following reason. If there are several energy saving clients, then each of them may choose an energy efficient plan which puts a greater burden on the server. This would overload the server, thereby, degrading the throughput of the entire distributed system.

Such a scenario can be avoided by placing a bound on the extra work that the server is

willing to perform in order to "help out" a client's energy problem. This is modeled by the server work threshold. Clearly, the *server work threshold must be a non-increasing function of the load at the server*, which is another novel issue introduced by the mobile computing environment.

Thus the optimal plan depends on the state of the client-server system. We approach this novel optimization criterion by compiling a query into multiple candidate plans, such that for every possible state of a client-server system, the optimal plan for that state is one of the candidate plans. When a query is submitted, the server chooses the plan from the sequence of candidate plans that is optimal for the current state.

Similar multi-criterion optimization criterion also arise in distributed databases. It may be argued that, in order to avoid placing a disproportionate amount of query processing work on a site that is already loaded, the choice of the optimal plan is a function of the load at the various sites. This leads to the following optimization criterion.

Minimize total resource utilization where $Res_i \leq Thresh_i$, where Res_i is the resource utilization at site i and $Thresh_i$ is a decreasing function of the load at site i. The idea of compiling queries into multiple plans for a given SQL statement such that the plan chosen at execution time depended on the load at that time was proposed in [2].

The techniques discussed in this paper can be applied to the above scenario as well, and, in general, to multi-criterion optimization problems in which each criterion is an *additive* metric. In this paper, we consider the problem of optimizing queries in a mobile computing environment as the typical example of our techniques.

Our contributions are as follows:

1. We present bounds on the maximum energy saved by choosing the least energy plan over the least work plan (which is the plan of choice in standard distributed databases [13, 30, 31]). When the server is lightly loaded, we show that the maximum savings is no more than a factor of $(\textit{idling coefficient})^{-1}$.

2. *Adaptation of Partial Order Dynamic Programming:* The basic observation is that partial order dynamic programming [17, 18] may be adapted to find all plans that could possibly be optimal for a certain state of the client-server system. We show how to adapt the partial order dynamic programming strategy (p.o. dp) to this scenario such that the *coverset* of plans [17, 18] returned by p.o. dp is the set of candidate optimal plans for a query.

3. *Linear Combinations Algorithm:* The coverset of plans plans obtained from a partial order dynamic programming is approximated by the *linearset* of the query. The linearset of a query, is the set of plans that are optimal for a linear combination of client work and server work (i.e., client work + $g\times$ server work, where g is a real constant). We present a novel 2-dimensional algorithm that computes the linearset of the query. Although, there are (uncountably) infinite number of linear combinations, the com-

plexity of our algorithm is $2l\times$ complexity of running a standard System R* optimizer, if the size of the linearset is l.

4. *Linearset Algorithm:* We present a novel, 2-dimensional dynamic programming technique that computes the linearset of a query.

5. *Accuracy of approximating coverset by linearset:* Simulation results show that the optimal plan returned by the linearset is the same as the optimal plan returned by the coverset technique (no exceptions were found in over a million runs). We *conclude that the linearset is a practical approximation of the coverset.*

6. *Performance comparison of search algorithms by simulation.* We show, by means of simulation, that the linearset algorithm and the linear combinations algorithm outperform the p.o. dp algorithm by factors of 1.6 to 8. The performance of the linear combinations algorithm and the linearset algorithm is quite comparable. The time taken by the linear algorithms to complete is at most twice the time taken by a standard System R* like search algorithm. *We conclude that in this scenario, either the linearset algorithm or the linear combinations algorithm should be used for the search algorithm.*

7. *Size of coverset and linearset.* The efficiency of the runtime choice of the optimal plan depends linearly on the size of the coverset or on the size of the linearset, depending on which of the two sets are computed. Our simulations show that the size of the linearset is small (within 12) and the size of the coverset is about 1.5 to 3 times the size of the linearset.

Based on the above, we propose that the linear search techniques are practical for optimizing multi-criterion objective function for additive metrics.

The rest of the paper is organized as follows. Section 2 discusses the alternatives of query initiation and proposes that the server initiation rule be used. Section 3 presents examples to show how energy may be saved by choosing alternate query plans. Section 4 discuses the cost model and the optimization criterion. In Section 5 we present bounds on the maximum benefit that can be incurred by optimizing for energy as opposed to the standard metric of total work [13, 30, 31]. Section 6 discusses the coverset, linearset and linear combinations algorithm. Section 7 presents the performance comparison. Finally, we conclude in Section 8.

2 Query Initiation Rules

There are two different ways in which query execution may be initiated. In this section, we present these two alternatives, called server initiation and client initiation and argue that server initiation avoids overloading the server.

The compilation of a query results in a sequence of plans such that for any load at the server and any energy level of the client, the optimal plan is one among the sequence of plans. Hence, the selection of the optimal plan from the pre-compiled sequence requires two values, (1) the energy level of the client, and (2) the load at the server. This gives the following two alternatives for query initiation:

1. *Client Initiation.* In this scenario, the server broadcasts its load periodically over a channel and the client selects the plan based on the broadcasted load and its energy level. The server is informed of the plan selected and it increases its load proportionately.

2. *Server Initiation.* The client sends its energy level to the server and the server selects the plan. Once a plan is selected, the load at the server is increased proportionately.

Consider a scenario when there are many clients ready to submit queries simultaneously and the server is lightly loaded. Since the server is lightly loaded, the server work threshold is high and hence, the optimal plan at light load may place a greater burden at the server. If each client makes the same local decision, this may result in overloading the server.

This problem is mitigated by server initiation. If multiple clients submit queries simultaneously, these are considered in some serial order by the server[1]. After each query is considered, the load of the server is increased to reflect the fact that a new query is now being executed. Hence, when a subsequent query is considered, the value of the load on the server is higher, resulting in the choice of a possibly different optimal plan. Hence, multiple clients do not "see" the same state of the server. The server initiation scheme avoids overloading the server due to multiple clients compiling queries into energy efficient plans simultaneously.

We show in Section 7.3.2 that the choice of the optimal plan from the compiled set of plans can be done quite efficiently. Hence, the overhead imposed by the server initiation rule on the server is not overwhelming. An alternative approach to the problem is to extend global query optimization techniques [34].

3 Examples

In this section, we present an example to show that the energy consumption of work optimal plans may be significantly higher (factors of 3-12) than the energy consumption of energy optimal plans.

Figure 1 depicts the power consumption of a typical state of the art portable computer, the AMD Saturn SXL [1]. Devices may be switched between modes under application control.

[1]The choice of an optimal plan from a sequence of plans can be done efficiently enough to be considered atomic.

Device	Mode	Power	Mode	Power	Mode	Power
Motherboard	Full On	4538 mW	Doze	1319 mW	Suspend	373 mW
Hard disk	Read/Write Seek	3000 mW	Idling	1045 mW	Suspend	0 mW
Display	Full on	2048 mW	Backlight off	608 mW	Dark Screen	0 mW

Figure 1: Typical Power consumption of various components in a computer

Assume there is a relation `STOCKS` residing at the server and a relation containing all entries in `STOCKS` for the month of `AUG92` cached at the client. Consider the following SQL query:

```
SELECT COMPANY-NAME
FROM STOCKS
WHERE PRICE > 50.0 AND MONTH = AUG92
```

Let us examine the following two plans to evaluate the query:

1. *Client Plan* (local evaluation). In this plan, the query is processed locally, and takes 1 second to process. For simplicity, we assume that 20% of the time is spent in the CPU and 80% on the disk.

2. *Server Plan* (remote evaluation). The query is *shipped* to the server and also takes 1 second to process at the server. An additional 1% overhead is incurred due to shipping the query and receiving the result.

Due to caching, the client is able to match the higher computing speed of the server by searching a much smaller volume of data than the server. Clearly, the client plan is the plan and the minimum work plan and the work done by the server in this plan is 0 seconds.

Figure 2 depicts the energy consumed (analytically calculated) by running the two plans under various modes. The server plan does not utilize the client disk at all, hence the disk may be turned off during query execution (clearly this is not possible for the client plan). Thus, the server plan may be evaluated with (a) the client disk ready but idle during remote operations, or (b) with the client disk suspended when no local processing is being done. Clearly, option (b) is not always practical for general queries, and in practice the energy consumed would lie between those of (a) and (b); both numbers are listed in Figure 2.

Even though the client plan is the optimal work plan [13, 30, 31], as Figure 2 shows, the server plan is significantly more energy-efficient, between factors of 3 to 10, depending

	Screen Full On		Screen Backlight off		Screen Dark	
	Disk Spin	Disk Susp.	Disk Spin	Disk Susp.	Disk Spin	Disk Susp.
Client Plan	6.54 Joules	n.a.	5.2 Joules	n.a.	4.5 Joules	n.a.
Server Plan	4.3 Joules	3.3 Joules	2.9 Joules	1.9 Joules	1.4 Joules	0.4 Joules
Idling Coefficient	0.66	0.50	0.56	0.37	0.31	.09
Energy Savings	33%	49.5%	44%	63%	69%	91%

Figure 2: Table comparing the energy consumptions of two plans for the same query in several modes

on the value of the idling coefficient. Clearly, the plan that would be chosen by a traditional query optimizer is not necessarily the energy-optimal one.

4 Cost Model and Optimization Criterion

In this section, we discuss the cost model for energy consumption and the optimization criterion.

4.1 Cost model

Let t_c be the total computation time spent on the client and t_s be the total computation time spent on the server. Let α denote the multi-programming delay factor at the server, i.e., if t_s is a measure of the computation time of a plan at the server, then $\alpha(L) \times t_s$ denotes the response time of the server when the load value is L. Clearly, $\alpha \geq 1$.

Without accounting for possible parallelism between client and server computation, we assume that the response time is $t_c + t_s \times \alpha$. Suppose that the client has a power consumption of P watts during normal operation mode. By definition of the idling coefficient, the power consumption in energy saving mode is $i \times P$ watts. The energy consumption by the client in a plan p, denoted by $E(p)$, may be divided into two parts:

1. Energy consumed by the client due to computation at the client, which is $t_c \times P$ joules.

2. Energy consumed by the client due to computation at the server, which is $t_s \times \alpha \times i \times P$ joules, since the client may switch to an energy saving mode during server computation.

Thus the energy consumption of a plan is given by $(t_c + t_s \times \alpha \times i) \times P$ joules. For a plan p, let $W_c(p)$ and $W_s(p)$ denote the resource utilization of the plan at the client and the server respectively. We assume further that the ratio of computation time to total resource utilization is almost the same at both the server and the client, i.e., $t_c/W_c = t_s/W_s$.

Hence, the energy consumption of a plan is proportional to (not equal to)

$$E(p) \equiv W_c(p) + i \times W_s(p) \times \alpha$$

and we use this as the cost function for energy. The work done by a plan p, denoted by $W(p)$ is defined as

$$W(p) = W_c(p) + W_s(p)$$

We assume that the estimation of W_c and W_s is done using a cost model similar to that used by System R* [13, 30, 31].

4.2 The Optimization Criterion

In this section, we discuss the optimization criterion.

Let L denote the load at the server and B denote the battery energy level of the client at a specific time. Suppose a query is initiated at that time. We propose that the objective function is:

Minimize $W_s(p)$ where $W_s(p) \leq h_{server}(L) \times W_{fair}$ and $E(p) \leq h_{energy}(B) \times E_{min}$, where

- W_{fair} denotes the server work done by the plan that is optimal for the metric $W_c + W_s$.
- E_{min} is the client energy consumption of the minimum energy $(= W_c + i \times \alpha \times W_s)$ plan.
- $h_{energy}(B)$ denotes the energy threshold and is (a) no less than 1 for all values of battery energy level B, and (b) is an increasing (or non-decreasing) function of B.
- $h_{server}(L)$ denotes the server work threshold and is (a) no less than 1 for all values of load L, and (b) is a decreasing (or non-increasing) function of L.

For a given value of the load L and client energy level B, the optimization criterion partitions the space of plans into two halves, a feasible space and its complement. The space is pictorially depicted in Figure 3.

We assume that during query initiation, the client sends the value of $h_{energy}(B)$ to the server and the server uses its values of α and $h_{server}(L)$ to choose the plan from the candidate set of plans. The values of E_{min} and W_{fair} are estimated during compilation.

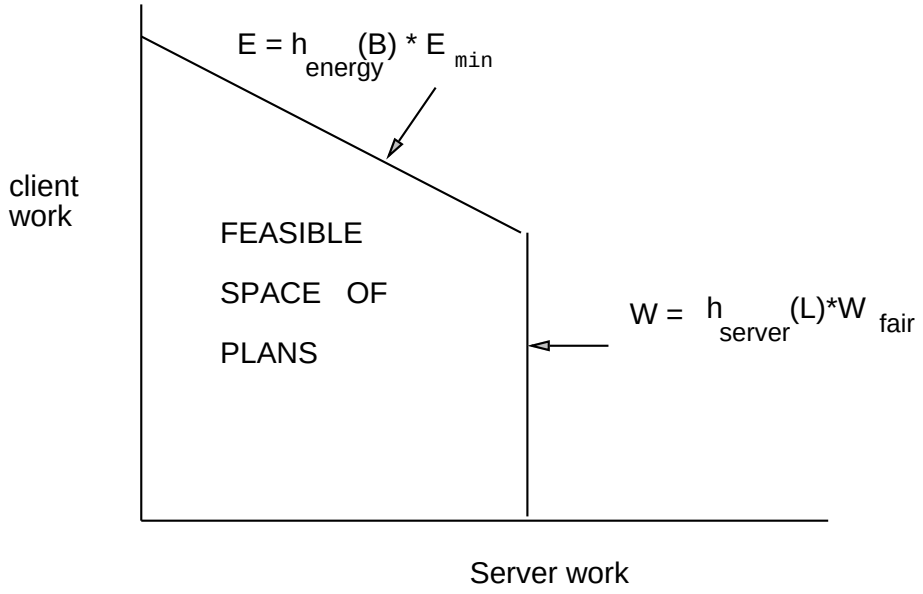

Figure 3: The optimization criterion and the feasible space of plans

5 Comparison of Work and Energy Metrics

In this section, we present a bound on the maximum energy savings that can result from optimizing for energy instead of work. The bound is derived in a very general setting without making any assumptions about the execution space of plans.

The following theorem shows that the energy savings obtained (i.e., the ratio of the energy consumed by the best work plan to the best energy plan) is bounded within a factor of $max(i \times \alpha, (i \times \alpha)^{-1})$.

Theorem 1: Let p_E denote the plan with the least energy and p_W denote the plan with the least work. Then,

$$\frac{E(p_W)}{E(p_E)} \leq max(i \times \alpha, \frac{1}{i \times \alpha})$$

To see the implication of this theorem, consider a lightly loaded server and a client with an idling coefficient of 0.6. Since the server is lightly loaded, $\alpha \approx 1$. Then, according to the above theorem, the maximum possible savings in energy is no more than 40% over the energy consumed by the least work plan. Since, actual savings are clearly less than the maximum savings, hence, in such a scenario, it is not productive to optimize for work and energy separately.

When the server is lightly loaded, the maximum energy savings is given by $1/i$, which for current machines ranges between 1.6-10. Hence, energy efficient query optimization is meaningful only in scenarios where the idling coefficient is small.

6 Search Algorithms

In this section, we discuss the design of search algorithms. We present three optimization techniques, namely, an adaptation of partial order dynamic programming, the linear combinations technique and the linearset technique.

The problem of optimizing queries in a mobile environment is different from standard database query optimization. The threshold functions for load, i.e., $h_{server}(L)$ and the threshold function for energy, i.e., $h_{energy}(B)$ are functions of the state of the client-server system. Hence, an optimal plans for a specific L and B value may no longer be an optimal plan when the load and energy levels change.

The basis of the solution lies in the following observation presented in Theorem 2, namely, that, for any state of the client-server system, the optimal plan for that state is a member of the coverset for the partial order $\leq_1$ defined below.

Definition 1: Let p_1 and p_2 be two plans for the same subquery and let i be the idling coefficient. We define $p_1 \leq_1 p_2$ if

1. $W_s(p_1) \leq W_s(p_2)$, and
2. $W_c(p_1) + i \times W_s(p_1) \leq W_c(p_2) + i \times W_s(p_2)$. □

Theorem 2: Let p be an optimal plan for the optimization criterion for a specific value of the load L and the energy level B. Then p is a member of the coverset of the partial order $\leq_1$. □

A partial order dynamic programming algorithm [17, 18] may be designed using the partial order $\leq_1$ and extending the System R* optimizer design.

6.1 Linearset of a Query

In this section, we discuss the concept of the linearset of a query, which is an approximation to the coverset. We first define the concepts of linearly optimal plans and the linearset of a query.

Definition 2: Given a real number $g \geq 0$, a plan p is said to be *linearly optimal* for the gradient g, if p has the smallest value of *client-work* $+ g \times$ *server-work* over the set of all plans. A plan p is said to be linearly optimal if there exists some gradient $g \geq 0$ for which

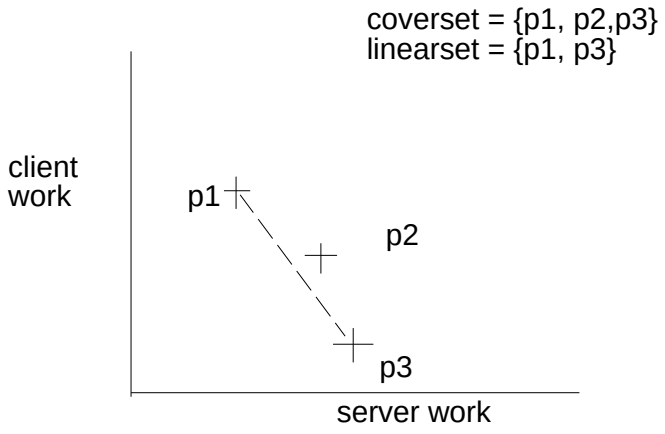

Figure 4: Coverset and linearly optimal plans

it is linearly optimal. The set of all linearly optimal plans for a query is called the *linearset* of the query. □

The relationship between linearset and coverset of a query is discussed in the following theorem, which states that every plan in the linearset is a member of the coverset. Thus the linearset is a subset of the coverset.

Theorem 3: Let p be a linearly optimal plan for $m \geq i$, where i is the idling coefficient. Then, p is a member of the coverset for $\leq_1$. □

However, the converse is not necessarily true, i.e., not every plan in a coverset is a member of the linearset. In particular, let p_2 be a plan that lies on the opposite side of the origin of the straight line that is obtained by joining two plans p_1 and p_3. The reader may convince himself/herself that p_2 is not linearly optimal. Figure 4 illustrates this point.

Hence the set of plans, which are linearly optimal for gradients $\geq i$, is an approximation to the coverset of a query with respect to the partial order $\leq_1$. The accuracy of this approximation is discussed in Section 7.1.

6.2 Linear Combinations Algorithm

In this section, we present the linear combinations algorithm for computing the linearset of a query.

The algorithm is explained by mapping each plan p for a query into a point (x, y) in the (W_s, W_c) space, i.e., two-dimensional space, where x represents $W_s(p)$ and y represents $W_c(p)$. The basis of this algorithm are the following observations:

Input: Points representing plans p_1 and p_2 such that p_1 is optimal for some gradient g_1 and p_2 is optimal for gradient g_2, where $g_1 \geq g_2$.
Output: Finds all plans that are linearly optimal for some gradient g such that $g_1 > g > g_2$.
$linear\_comb(p_1, p_2)$
begin
    if $(p_1 == p_2)$ return NIL;
    g_3 = gradient of the line joining p_1 and p_2.
    $p_3 = optimize(g_3)$;
    if $((p_3 == p_1)$ or $(p_3 == p_2))$ **return** NIL;
    else **return** $linear\_comb(p_1, p_3) \circ p \circ linear\_comb(p_3, p_2)$;
end.

Figure 5: Algorithm 1: The Linear Combinations Technique

1. if x and y are any two *additive* metrics, (i.e., $x(p \bowtie q) = x(p) + x(q)$, for subplans p and q), then $y + g \times x$,s also an additive metric, for any real number g.

2. Additive metrics satisfy the principle of optimality and can be optimized using a standard System R like dynamic programming algorithm.

Standard resource utilization metrics, such as W_c, W_s are additive metrics. The problem is to find all the linearly optimal plans without enumerating the set of all linear combinations (which is uncountably infinite). The algorithm is based on Theorem 4 and is presented in Figure 5. It uses the following two procedures:

1. For a real number g, $optimize(g)$, invokes a System R* like distributed database optimizer and returns the plan with the least value of $W_c + g \times W_s$.

2. Given lists of plans L and M, $L \circ M$ returns the concatenation of L and M.

The algorithm should be invoked using $linear\_comb(optimize(i), optimize(\infty))$, where i is the idling coefficient. The recursive step of the algorithm is geometrically illustrated in Figure 6. The basis of the linear combinations algorithm is Theorem 4.

Theorem 4: Let p_1, p_2 and p_3 be linearly optimal plans for gradients g_1, g_2 and g_3, where $g_1 > g_2$. Then, $g_1 > g_3 > g_2$ if and only if p_3 lies within the triangle bounded by the following three lines:

1. the line with gradient g_1 passing through p_1.

2. the line with gradient g_2 passing through p_2.

3. the line joining p_1 and p_2. □

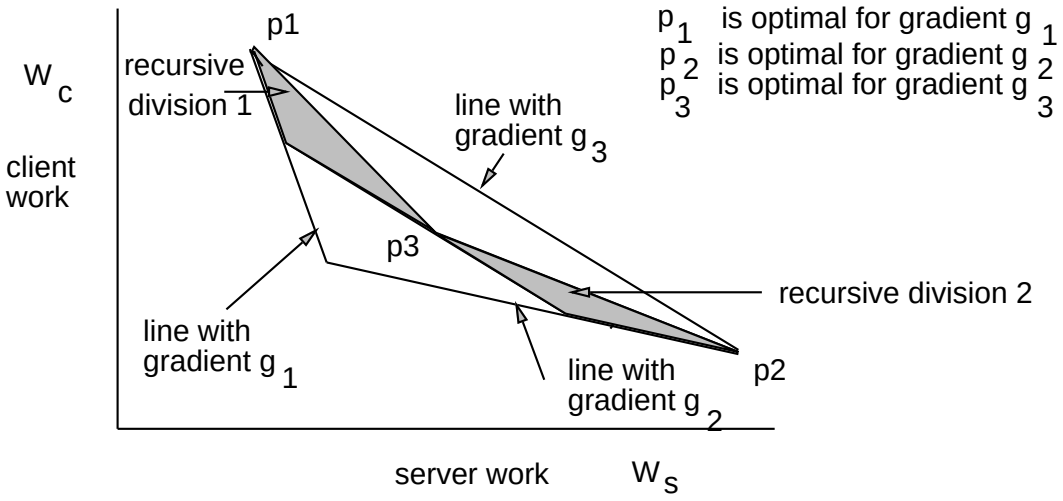

Figure 6: The linear combinations technique: divide and conquer algorithm

Theorem 5 proves that the number of invocations of the standard R* like optimizer algorithm as part of the execution of the linear combinations algorithm is bounded by $2l$, where l is the size of the linearset of the query.

Theorem 5: Suppose that the size of the linearset of a query is l. Then, the number of times the procedure *optimize* is invoked is no more than $2l$. □

6.3 The linearset algorithm

In this section, we present an alternative algorithm for computing the linearset of a query, called the *linearset algorithm*.

The *linearset algorithm* is based on the following simple observation.

The linearset of a query is obtained by extending the linearsets of subqueries of the query.

Hence, a dynamic programming approach may be used to compute the linearset of a query. This algorithm proceeds as follows: For each subquery, we compute the linearset of the subquery. Assume that inductively, the linearset of all subqueries of size $n-1$ have been computed. For each query Q of size n, we consider all subqueries S of the query Q of size $n-1$. For each subquery S, we extend the linearset of S to obtain a set of candidate plans for Q. Finally, we compute the linearset of the set of candidate plans (i.e., merge the linearsets of the extended subqueries) to obtain the linearset of the query Q.

7 Performance Evaluation

In this section, we experimentally evaluate how well a linearset approximates the coverset for a query. We then compare the performance of the three algorithms discussed, namely, the coverset, linearset and linear combinations algorithm. Finally, we discuss the performance of run time selection of the optimal plan from the linearset and the coverset.

7.1 Approximating Coverset by Linearset

In this section, we discuss results of simulations designed to measure the error of approximating the coverset by a linearset.

Simulations were performed by computing the coverset for a query on n relations. The partial order dynamic programming algorithm proceeds by extending subplans by one join to generate a plan for a larger subquery. Since there could be multiple join methods and access methods, there are multiple possible extensions, each with a possibly different cost, for each subquery, query pair.

In our simulation, we modeled the cost of each extension as a random number in the W_c, W_s space. The number of extensions was varied between 3 and 10. In an actual experiment, the random estimation of costs would be replaced by calls to the cost model. An implementation of a cost model similar to that used by the System R* is under progress and will be used to verify the conclusions of the simulations [19].

The linearset for the query is computed from the coverset. For a choice of the parameters (i.e., idling coefficient, h_{energy}, h_{server} and delay factor α) that define the state of the client-server system, we choose the optimal plan from the coverset and from the linearset and compare their values for server work.

The idling coefficient was varied between 0.2 and 0.6. The parameters h_{energy} and h_{server} were varied between 1 and 100. The parameter α was varied between 1 and 20. The number of simulations performed for each state was statistically significant.

The result of the simulation is summarized below: *For all values of idling coefficient, h_{server}, h_{energy}* and α, *the optimal plan from the coverset was found to be identical to the optimal plan from the linearset.*

In other words, although the linearset is an approximation of the coverset, the simulation did not find a single case in which the optimal coverset plan was different from the best linearset plan. We conclude that the *linearset is a good approximation to the coverset* for the mobile computing scenario.

7.2 Comparison of coverset, linearset and linear combination algorithms

The simulation proceeded by generating a random set of k extensions for plans for each query, subquery pair as explained in the Section 7.1. The parameter k was varied between 3 and 10, and for a particular run, was kept constant. For each value of the parameter, i.e., for each run, the three algorithms are executed a (statistically) large number of times. The running times of each of the algorithms was measured and plotted. The entire experiment was repeated by varying the number of relations in the query from n =3 to 9.

Figure 7, 8 and 9 show the performance of each of the three algorithms as measured by the simulation, when $k = 3, 5$ and 10. For reference, the performance of a System R* like search algorithm was also simulated. These simulations were run on a Sparc II workstation with 16 MB of main memory and 95 MB of swap memory.

Our observations are summarized below.

1. The *linear_comb* and *linearset* algorithms consistently perform better than the *coverset* algorithm by factors ranging from 1.6 to 8.
2. The performance of *linear_comb* and *linearset* is comparable, although, *linear_comb* seems to perform better as the number of extensions is increased.
3. The time taken to complete the linearset and the linear combination algorithms is within a factor of 2 of the time taken to complete a System R* like standard search algorithm.

We conclude that the linear_comb algorithm may be used as a practical choice for query compilation.

A detailed performance evaluation using the cost model of System R* is currently under implementation in [19].

7.3 Runtime selection of the optimal plan

In this section, we discuss the algorithm to select the optimal plan at run time. We also measure the size of the linearset and coverset of a query using a simulation, since they determine the performance of the run time selection algorithm.

7.3.1 Algorithm for runtime selection

At runtime, when a query is submitted, the optimal plan is selected from the set of candidate plans. Clearly, the selection procedure must be efficient.

Figure 7: Performance Comparison of the linearset, linear_comb and coverset algorithms

Figure 8: Performance Comparison of the linearset, linear_comb and coverset algorithms

Figure 9: Performance Comparison of linearset, linear_comb and coverset algorithms

We assume that the compilation procedure generates the following.

1. A set of plans, that is kept sorted in increasing order of server-work W_s.

2. The value of W_{fair}, which is the server work done by the plan that has the least total work.

3. The value of E_{min}, which is the value of the energy consumed by the plan that consumes the minimum energy.

At runtime, the values of the delay factor α, idling coefficient i, server work threshold h_{energy} and client energy threshold h_{energy} are known. The optimal plan may be selected as follows.

Scan the sorted list of plans in sequence. Return the *first plan* that satisfies the energy and work constraints.[2]

Hence the algorithm scans the list of plans at most once. This algorithm would be practical if the size of the coverset and/or the linearset is not too large.

7.3.2 Size of the coverset and linearset

In this section, we present results of simulation designed to measure the sizes of the coverset and the linearset of a query.

The simulation strategy used is similar to the ones used for the earlier experiments in this section. The number of relations n in a query are varied from 3 to 9 and the cost of extending a subquery is chosen to be a random number in the W_c, W_s space. The number of extensions, k for every query, subquery pair is varied between 3 and 10. The size of the coverset and the linearset are measured for each run and a statistically large number of runs are executed for different values of n and k.

Figure 10 presents the results of the simulation for three values of k, namely, $k = 3, 5$ and 10. Our observations are summarized below:

1. The size of the covereset is about 1.5 to 3.5 times the size of the linearset.

2. The size of the linearset of a query is small enough to result in a practical runtime selection algorithm[3].

We conclude that the run time overhead incurred by computing the linearset of plans is quite small.

[2] In a coverset or a linearset, if plans are sorted in increasing order of W_c, they are automatically sorted in decreasing order in the other axis, namely, W_s.

[3] On the average, the run time algorithm scans a list of length ≤ 14, which for practical purposes can be considered atomic.

Figure 10: Size of linearset and coverset

8 Conclusions

The paper presents three general techniques for solving multi-criterion query optimization problems. Such optimization criterion arise naturally in mobile computing environments, where the appropriate criterion considers both resource utilization and energy consumption, and in distributed database environment, where the choice of an optimal plan may depend on the load at the various sites.

A performance analysis by simulation shows that the linearset and linear combinations algorithm are a practical choice of search algorithm for such problems.

The techniques presented in the paper can be used to optimize ad-hoc queries, i.e., queries that are executed only once for a given state of the client-server system. Optimization of ad-hoc queries is discussed in [3, 4].

Work is in progress [19] to validate the simulation results obtained using more accurate experiments. Future work includes extending our two dimensional algorithms to three and higher dimensional spaces.

Acknowledgements

We thank Tony Dahbura, Peter Honeyman, and Brian Marsh at MITL, Panasonic, for their help in gathering power consumption information. Hank Korth at MITL, Panasonic, observed and suggested the applicability of the ideas in our paper to general distributed environments. We thank B. Badrinath and Tomasz Imielinski at Rutgers University, Narain Gehani at AT&T Bell Labs and Avi Silberschatz[4] at University of Texas, Austin for interesting conversations on the subject. Finally, we thank the anonymous referees for their comments.

References

[1] Advanced Micro Devices, *Saturn SXL Notebook Computer Functional Product Specification*, Release Date 6/25/91, Revision 1.01.

[2] R. Alonso. "Query Optimization in Distributed Databases Through Load Balancing" *PhD Dissertation*, Division of Computer Science, University of California, Berkeley, 1986.

[3] R. Alonso and S. Ganguly. "Query Optimization for Energy Efficiency in Mobile Environments", In *Proceedings of the 1993 International Workshop on Foundations of Models and Languages for Data and Objects*, Aigen, Austria.

[4] currently at AT&T Bell Labs, on leave from UT, Austin.

[4] R. Alonso and S.Ganguly. "Energy Efficient Query Optimization" In *MITL Technical Report 33-92*, Nov. 1992.

[5] R. Alonso, E. M. Haber, and H. J. Korth. "A Database Interface for Mobile Computers", In *Proceedings of the 1992 Globecomm Workshop on Networking of Personal Communications Applications*, Orlando, Florida.

[6] R. Alonso and H. J. Korth. "Database System Issues in Nomadic Computing", *Proceedings of the 1993 ACM-SIGMOD Conference on the Management of Data.*

[7] P.M.G. Apers, A.R. Hevner and S.B. Yao. "Optimization algorithms for Distributed Queries", *IEEE Transactions on Software Engineering*, Volume SE-9, No 1,Janu ary 1983).

[8] B.R. Badrinath and T. Imielinski. "Replication and Mobility", *Proceedings of the 1992 Workshop on the Management of Replicated Data*, Monterrey, California.

[9] D.S. Batory. " Concepts for a Database System Synthesizer", In *Proceedings of the 1988 ACM Symposium on Principles of Database Systems.*

[10] R.E. Bellman. *Dynamic Programming*, Princeton University Press, 1957.

[11] P.A. Bernstein and D.M. Chiu. "Using Semi-Joins to Solve Relational Queries", *Journal of the ACM*, **28**:1, 1981.

[12] W.W. Chu and P. Hurley. "Optimal Query Processing for Distributed Systems", *IEEE Transactions on Computers*, **C-31**:9, Sept. 1982.

[13] D. Daniels, "Query Compilation in a Distributed Database System", *IBM Technical Report RJ3423*, 3/22/82.

[14] Eager. "Advances in Rechargeable Batteries Pace Portable Computer Growth", In *Proceedings of the 1991 Silicon Valley Personal Computer Conference.*

[15] R. Epstein, M. Stonebraker and E. Wong. "Distributed Query Processing in a Relational Database System", *Proceedings of the 1980 ACM SIGMOD Conference on Management of Data.*

[16] J.C. Freytag. "A rule-based view of query optimization", In *Proceedings of the 1987 ACM-SIGMOD Conference on the Management of Data.*

[17] S. Ganguly, W. Hasan and R. Krishnamurthy. "Query optimization for Parallel Executions", In *Proceedings of the 1992 ACM-SIGMOD Conference on Management of Data.*

[18] S. Ganguly. "Parallel Evaluation of Deductive Database Queries", *PhD Dissertation, The University of Texas at Austin*, August 1992.

[19] A. Goel. "Multi-Criterion Query Optimization", *Masters Thesis in Preparation*, Department of Electrical Engineering, Rutgers University.

[20] N. Goodman, P.A. Bernstein, E.Wong, C.L. Reeve and J.B. Rothnie. "Query Processing in SDD-1: A System for Distributed Databases," *ACM Transactions on Database Systems.*

[21] G. Graefe and W.J. McKenna. "The Volcano Optimizer Generator:Extensibility and Efficient Search", In *Proceedings of the 1993 IEEE Conference on Data Engineering.*

[22] GRID, *GRIDPAD Radio Module Specifications*, 1992.

[23] W. Hong and M. Stonebraker, "Optimization of Parallel Query Execution Plans in XPRS" *Proceedings of the 1991 International Conference on Parallel and Distributed Information Systems*

[24] T. Imielinski and B. R. Badrinath. "Querying in Highly Mobile Distributed Environments", *Proceedings of the 1992 Very Large Database Systems Conference.*

[25] T. Imielinski and B. R. Badrinath. "Data Management for Mobile Computing", *Sigmod Record*, March 1993.

[26] Y.E. Ioannidis and Y.C. Kang. "Randomized Algorithms for Optimizing Large Join Queries", In *Proceedings of the 1990 ACM-SIGMOD Conference on Management of Data.*

[27] Y.E. Ioannidis and Y.C. Kang. "Query Optimization by Simulated Annealing". In *Proceedings of the 1987 ACM-SIGMOD Conference on Management of Data.*

[28] M. Jarke and J. Koch. "Query Optimization in Database Systems", In *ACM Computing Surveys*, **18**(2), June 1984.

[29] R. Krishnamurthy, H. Boral, and C. Zaniolo, "Optimization of Nonrecursive Queries", In *Proceedings of the 1986 Very Large Database Systems Conference.*

[30] G.H. Lohman et. al. "Query Processing in R*", In **Query Processing in Database Systems**, (Kim, Batory and Reiner(eds.), 1985), Springer-Verlag, Hiedelberg.

[31] L.F. Mackert and G.M. Lohman. "R* Optimizer Validation and Performance Evaluation for Local Queries", *Proceedings of the 1986 Very Large Database Systems Conference.*

[32] C. O'Malley. "The SL Germination - Comparison Report" *Pen, The Magazine of Pen-based Computing* Issue 7, May/June 1992.

[33] P. Selinger et. al. "Access Path Selection in a Relational Database Management System", In *Proceedings of the 1979 ACM SIGMOD Conference on Management of Data.*

[34] T.K. Sellis. "Global Query Optimization", In *Proceedings of the 1986 ACM SIGMOD Conference on Management of Data.*

[35] A. Swami and A. Gupta. "Optimization of Large Join Queries", In *Proceedings of the 1988 ACM SIGMOD Conference on Management of Data.*

[36] A. Swami. "Optimization of Large Join Queries", *PhD Thesis, Stanford University 1989.*

[37] E. Wong and K. Youssefi, "Decomposition —a Strategy for Query Processing", *ACM Transactions on Database Systems*,**1**,3, September 1976.

Appendix A

In this section we present the proof of Theorem 1.

Proof: The cost metric for energy is proposed to be

$$E(p) \equiv W_c(p) + i \times \alpha W_s(p)$$

Let β be the constant of proportionality, i.e., let

$$\beta E(p) = W_c(p) + i \times \alpha W_s(p)$$

Let γ denote $i\alpha$.

1. Case 1: $\gamma \leq 1$. Then

   $$\gamma W(p) \leq \beta E(p) \leq W(p)$$

   Hence

   $$\frac{E(p_W)}{E(p_E)} = \frac{\beta E(p_W)}{\beta E(p_E)} \leq \frac{W(p_W)}{\gamma W(p_E)}$$

   Since $W(p_E) \geq W(p_W)$,

   $$\frac{E(p_W)}{E(p_E)} \leq \frac{1}{\gamma}$$

2. Case 2: $\gamma > 1$. Then

   $$\gamma W(p) \geq \beta E(p) \geq W(p)$$

   Hence

   $$\frac{E(p_W)}{E(p_E)} = \frac{\beta E(p_W)}{\beta E(p_E)} \leq \frac{\gamma W(p_W)}{W(p_E)} \leq \gamma$$

Hence, in both cases,

$$\frac{E(p_W)}{E(p_E)} \leq max(\gamma, \frac{1}{\gamma}) \quad \square$$

Appendix B

In this section, we present the proof of Theorem 2.

Proof: Suppose that p is an optimal plan for some specific value of the load L and energy level B but is not a member of the coverset of $\leq_1$. Then, there exists a plan q, such that $W_s(q) < W_s(p)$ and $W_c(q) + i \times W_s(q) < W_c(p) + i \times W_s(p)$.

Since, p satisfies the work threshold so does q. We now show that q satisfies the energy threshold as well. Note that $\alpha \geq 1$, for all values of the load L.

$$\begin{aligned} E(q) &= W_c(q) + i \times \alpha W_s(q) \\ &= W_c(q) + i \times W_s(q) + i \times (\alpha - 1) W_s(q) \\ &< W_c(p) + i \times W_s(p) + i \times (\alpha - 1) W_s(p) \\ &= E(p) \end{aligned}$$

Hence, q has a smaller value of the objective function W_s, and lies within the feasible space. Hence, p is not optimal for the value of load L and energy level B, contradicting the hypothesis. $\square$

Appendix C

In this section, we present the proof of Theorem 3.

Proof: Suppose that p is not a member of the coverset for $\leq_1$. Then, there exists a plan q such that $W_s(q) < W_s(p)$ and $W_c(q) + i \times W_s(q) < W_c(p) + i \times W_s(p)$. Then, for $m \geq i$,

$$\begin{aligned} & W_c(q) + m \times W_s(q) \\ = \; & W_c(q) + i \times W_s(q) + (m-i)W_s(q) \\ < \; & W_c(p) + i \times W_s(p) + (m-i)W_s(p) \\ = \; & W_c(p) + m \times W_s(p) \end{aligned}$$

which means that p is not a linearly optimal plan for $m \geq i$. □

Appendix D

In this section, we present the proof of Theorem 4.

Proof: Suppose $g_1 > g_3 > g_2$. Let $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$ and $p_3 = (x_3, y_3)$ in the W_s, W_c space. Let g be the gradient of the line joining p_1 and p_2. Define regions R_1, R_2 and R as follows:

$$R_1 = \{(x, y) \mid y + g_1 x < y_1 + g_1 x_1\}$$
$$R_2 = \{(x, y) \mid y + g_2 x < y_2 + g_2 x_2\}$$
$$R = \{(x, y) \mid y + gx < y_1 + gx\}$$

We make the following observations.

1. There are no plans in the region $R_1 \cup R_2$, since p_1 and p_2 are optimal for gradients g_1 and g_2 respectively.
2. $y_1 + gx_1 = y_2 + gx_2$.
3. $y_1 \geq y_2$ and $x_1 \leq x_2$.
4. $g_1 \geq g \geq g_2$. To see this, suppose that $g_1 < g$. Then, p_2 has a lesser value of $y + g_1 x$, contradicting the fact that p_1 is optimal for g_1. Similarly, the case when $g_2 > g$ may be argued.

Let $(x, y) \in R - (R_1 \cup R_2)$.

Case 1: $g \geq g_3 \geq g_2$.

$$\begin{array}{lll} y + gx & \geq & y_2 + g \times x_2 \quad \text{since } (x, y) \in R \\ y + g_2 x & \geq y_2 + gx_2 & \text{since } (x, y) \notin R_2 \end{array}$$

Hence, for all g_3, such that $g \geq g_3 \geq g_2$

$$y + g_3 x \geq y_2 + g_3 x_2$$

Case 2: Suppose $g \leq g_3 \leq g_1$.

$$\begin{array}{ll} y + gx & \geq y_1 + gx_1 \quad \text{since } (x, y) \in R \\ y + g_1 x & \geq y_1 + gx_1 \quad \text{since } (x, y) \notin R_1 \end{array}$$

From both these cases, we conclude that if $(x, y) \in R - (R_1 \cup R_2)$

$$\forall g_3, g_1 \geq g_3 \geq g_2 \quad y + g_3 x \geq min(y_1 + g_3 x_1, y_2 + g_3 x_2)$$

Hence the plan that is linearly optimal for g_3, denoted by p_3, does not lie in $R - (R_1 \cup R_2)$. Hence, p_3 lies in the complement of this region, i.e., in $R^c \cup R_1 \cup R_2$ where R^c is the complement region of R. However, there are no plans in the region $R_1 \cup R_2$. Hence, the optimal plan lies in $R^c - (R_1 \cup R_2)$ which is triangular region discussed in the theorem statement.

The converse of the theorem is straightforward. □

Appendix E

In this section, we present the proof of Theorem 5.

Proof: We first establish the following statement. If the size of $linear\_comb(p_1, p_2)$ is m, then the number of times the procedure *optimize* is invoked in computing $linear\_comb(p_1, p_2) \leq 2m + 1$.

We prove this statement by induction. Let $n_{opt}(p_1, p_2)$ denote the number of invocations of the procedure *optimize* in the computation of $linear\_comb(p_1, p_2)$.
Base Case: $m = 0$. Clearly, $n_{opt} = 1 = 2m + 1$.

Induction Case:

$$n_{opt}(p_1, p_2) = 1 + n_{opt}(p_1, p) + n_{opt}(p, p_2)$$

$$m(p_1, p_2) = 1 + m(p_1, p) + m(p, p_2)$$

By induction hypothesis, $n_{opt}(p_1, p) \leq 2m(p_1, p) + 1$ and $n_{opt}(p, p_2) \leq 2m(p, p_2) + 1$. Hence

$$\begin{aligned} n_{opt}(p_1, p_2) &\leq 1 + 2m(p_1, p) + 1 + 2m(p, p_2) + 1 \\ &= 1 + 2(1 + m(p_1, p) + m(p, p_2)) \\ &= 1 + 2m(p_1, p_2) \end{aligned}$$

Suppose that l is the size of the linearset. If l_{opt} denotes the number of calls to the optimizer, then

$$l_{opt} = n_{opt}(optimize(0), optimize(\infty)) + 2$$

If $optimize(0) = optimize(\infty)$, then $l_{opt} = 2$ and $l = 1$. Otherwise

$$l = m_{opt}(optimize(0), optimize(\infty)) + 2$$

Hence

$$\begin{aligned} l_{opt} &\leq 3 + 2m_{opt}(optimize(0), optimize(\infty)) \\ &\leq 2l - 1 \ \square \end{aligned}$$