Energy efficient query optimization in nomadic computing

Report by: Liviu Teris, Ajaz Shaik

1. Motivation

Mobile devices and wireless networks are available to a large number of users nowadays; since the storing capacity and the processing power of mobile computers are close to what is expected from desktop devices, the possibility of hosting a distributed database on a network of such computers is brought into discussion.

2. Nomadic computing

Nomadic computing describes a paradigm in which we are dealing with a set of mobile computers interconnected via some wireless network and possibly communicating with some non-mobile servers. Mobile devices operate as part of a distributed system and are subjected to special constraints. These constraints are mostly caused by special hardware in the mobile devices, they set the differences from classical distributed systems and call for some extra consideration when porting a distributed database on a nomadic computing environment.

The most important aspects to be taken into account when dealing with such a system are:

- *Wireless networking* can constitute a bottleneck. Also, due to the fact that hosts are mobile and that they may operate in an environment with some amount of channel pollution, frequent disconnections are highly likely to occur. Although frequent, these disconnections are foreseeable – a change in the strength of the signal could announce an imminent disconnection and the computer may choose to download data in advance in order to process some query or declare itself "down" if it was participating in some voting protocol.
- *Reliability* can be an issue even more than in classical distributed system – due to the mobile nature of the devices (a catastrophic failure can be caused by dropping the device). To cope with such situations, the mobile device may for instance choose to transfer its logs to some non-mobile host.
- The issue of *limited battery capacity* is specific to nomadic computing. The battery life of a commercially available laptop is roughly 2 hours – this number

has remained more or less constant for the past 20 years. This is because even if there are constant improvements in battery technologies and in low power architectures, there is also a constant demand for even more powerful (and power consuming) CPU's, as users expect the performances of their mobile devices to come close to those of desktop machines. Also, whatever power gets saved is used for other purposes, such as better user interfaces. The approach that would help cope with power limitations is to change the software rather that the hardware; in the case of hosting a database, this translates to optimizing operations such as searches and updates, for power consumption.

3. Database issues in nomadic computing

The problem at hand is to host a distributed database on a nomadic computing system – two of the specific aspects that need to be accounted for are:

- *Communication costs*, both in terms of financial expenses, but also in what battery consumption is concerned. One aspect that can cause problems when designing an optimization algorithm is the asymmetrical character of wireless communication (for wireless devices it is cheaper energy wise to receive than to transmit the same amount of data).
- The overall *power consumed during query execution*; in order to account for this constraint, a new approach on query optimization algorithms may be necessary since, as will be discussed in the following paragraphs, the problem is not as trivial as adapting throughput oriented optimizers to work with some new energy metric.

4. Energy efficient query processing

This section discusses the design of a power aware query optimization algorithm to be used in nomadic computing. We are dealing with a system containing mobile clients and fixed servers that need to exchange data to complete queries. This mode of operation generates a conflict between the energy conscious clients and the throughput maximizing servers (i.e. the client would like to move as much of the computations on the server side whereas the server would like to only provide the client with the data it needs and complete its part of the query processing as quickly as possible).

Thus the goal of the optimizer is to find the query plan that satisfies both these individual goals to a given extent – ideally the client would save as much energy as possible so that the performance at the server side doesn't degrade beyond a given threshold. For the following discussion we will assume that we are dealing with a single

client communicating with a server, relations are fragmented, data can be stored at both the client and the server and the query is initiated by the client.

Classical query optimizers in distributed databases are usually concerned with maximizing throughput. Adding energy consumption to the scenario calls for a redesign of the optimization algorithm, since now we need to optimize along two axes (energy and time). In order to build the new optimizer the four elements of any query optimization system need to be defined, namely *the execution space, the cost model, the optimization criteria* and *the optimization algorithm*.

4.1 The execution space defines the search space for the algorithm. In this case it consists of the set of al query plans for a given query operation. These plans are described as annotated join trees where each internal node represents a join operation and the leaves represent base relations (the operations in a query can be rearranged and they can be performed at both the client and the server so that, for each query, a set of trees of different shapes describes the possible execution plans). In this case the annotations represent the cost of having computed the operations in a subtree i.e. the cost of a node is defined recursively in terms of the left and the right subtree. This way of defining the cost is convenient because it is easy to define but also because it is suitable for dynamic programming optimization algorithms.

4.2 The cost function characterizes the performance of the query plan as a combination of total work (expressed as total time spent in the query) and energy consumed. In order to be able to compare these two components, energy is described as a product of the power dissipated by each component (CPU plus memory, disk and peripherals) and the time spent on these components.

Devices	Power consumed
CPU and Memory	2 Watts
Disk	3 Watts
Constant power dissipation	4.6 Watts

Using these values the energy consumed at the client can be expressed as:

$$energy(p) = 2 * t_{cpu}^{client} + 3 * t_{disk}^{client} + 0.7 * t_{receive}^{client} + 1.5 * t_{send}^{client} + 4.6 * t_{rsp\_time}$$

And the total work as:

$$work^{server}(p) = t_{cpu}^{server} + t_{disk}^{server} + t_{send}^{server} + t_{receive}^{server}$$

$$work^{client}(p) = t_{cpu}^{client} + t_{disk}^{client} + t_{send}^{client} + t_{receive}^{client}$$

$$work(p) = work^{client}(p) + work^{server}(p)$$

The total response time that is part of the expression for energy is known to be non additive in distributed databases – in order to cope with that and make this metric suitable for a dynamic programming algorithm, the response time can be replaced by work in the energy function:

$$energy(p) = 2 * t_{cpu}^{client} + 3 * t_{disk}^{client} + 0.7 * t_{receive}^{client} + 1.5 * t_{send}^{client} + 4.6 * work(p)$$

The validity of this change is supported by the following inequalities:

$$t_{active}^{server}(p) + t_{active}^{client}(p) \geq t_{resp\_time}(p) \geq \max(t_{active}^{server}(p), t_{active}^{client}(p)) \geq (t_{active}^{server}(p) + t_{active}^{client}(p))/2$$

Thus

$$work(p)/2 \leq t_{resp\_time}(p) \leq work(p)$$

The error introduced by this approximation is acceptable especially when noticing that the main purpose of query optimization is not necessarily to find the best plan, but mainly to avoid the worst plan. This metric has a property called *additivity*, which as discussed further is desirable in optimization algorithms.

4.3 The third component of the optimization system is the **optimization criteria**. In this case the system needs to find the plan that consumes the least energy at the client and at the same time doesn't degrade the throughput at the server beyond a given point. Optimizers for total work are commercially available (*System R* for example); such an optimizer can be easily modified to optimize for energy alone (by plugging in the energy metric instead of that for work). Given such an optimizer we denote the work carried out in the minimal work plan as W_0. With this notation the optimization criterion for the new energy and work optimizer can be expressed as:

$$\text{cost}(p) = \begin{cases} energy(p) \text{ if } work(p) < k * WW_c(Q) \\ \infty \text{ otherwise} \end{cases}$$

i.e. find the minimal the plan with the minimal energy such that the total work doesn't increase by more than *k* times, where k is provided by the system administrator.

4.4 The fourth component of the optimization system is the **optimization algorithm** itself. The cost function mentioned above obeys the so called *principle of optimality* which states that: given a metric *m,* two plans *p1, p2* for a given subquery and an extension *e*, then

$$m(p_1) \leq m(p_2) \Rightarrow m(p_1 \circ e) \leq m(p_2 \circ e)$$

Additive metrics in general satisfy this principle – the work and the energy metrics described above are additive and thus each of them satisfies the principle of optimality. Metrics that abide by this principle can be used with dynamic programming algorithms since they allow for a ranking of the partial solutions at each iteration step.

Since we are dealing with optimizing along two dimensions and we have additive cost metrics for both dimensions, a dynamic programming algorithm with partial orders can be designed. That is, the algorithm maintains a set of incomparable but optimal plans for subqueries; at each step extensions are added to these subqueries until plans for the entire query are built.

In order to estimate the cost for each available extensions, terms of the following form are computed:

$$WW_r(S) = WW_c(Q) - WW_c(S)$$
$$EE_r(S) = EE_c(Q) - EE_c(S)$$
$$EW_r(S) = EW_c(Q) - EW_c(S)$$
$$WE_r(S) = WE_c(Q) - WE_c(S)$$

where the term $EW_c\ (S)$ denotes the energy consumed in the best work plan for subquery S - for each term, the first letter stands for what is measured, the second letter marks either the best work or the best energy plan, *c* stands for "consumed" and *r* for "remaining".

The algorithm iterates through 2 phases. In a first phase it runs the work and the energy optimizer to compute the above mentioned terms and in the second phase it adds these extensions to the subqueries in the optimal subset of plans with cardinality I in order to build the optimal subset for plans of cardinality i+1.

4.5 The problem with this raw search is that the solution space may be huge. In order to cope with this issue some pruning criteria need to be stated. For the optimization criteria stated in section 4.3, the following pruning criteria can be formulated:

- *Partial order:* if *work(p1) ≤ work(p2) and energy(p1) ≤ energy(p2)* then p2 can be pruned in favor of p1
- *Right ceiling:* if *work(p) > k*WW_c (Q)- WW_r (S)* then there is no sense in further exploring plan p – even if the best work extension is chosen the total work will go beyond the given threshold
- *Total order on energy:* if *energy(p_1) ≤ energy(p_2)* and *work(p_1) ≤ k* WW_c (Q) – WE_r (S)* then p2 will be pruned in favor of p1 because, even if the best energy extension is chosen, p1 will still have a total work that remains under the threshold.
- *Upper ceiling*: if *energy(p_2) + EE_r (S) ≥ energy(p_1) + EW_r (S)* and *p_1 is within the right ceiling,* then p2 can be pruned in favor of p1 (even if the best work extension is chosen, p1 will perform better energy wise than any extension of p2)

With these pruning criteria, the algorithm runs in 2 successive stages until a plan for the whole query is built:

- Obtain the cost for every extension for each of the plans in the partially ordered set
- Add extensions to the current plans, using the four pruning criteria to remove non promising plans.

5. Conclusion

The task of hosting a distributed database on a nomadic computing system comes with certain challenges – in this report the problem of having to cope with the limited battery life of the mobile devices in the network was discussed. The algorithm described above comes as a natural, although nontrivial, extension of the classical one dimensional query optimizers (optimizing for throughput only) in order to accommodate a second dimension to the search space, that of energy consumption.

6. Evaluation and extensions

The most valuable feature of the algorithm described above is that it is generic, in the sense that it can be applied to any multi dimensional optimization problems as long as additive cost metrics for each dimension are defined.

On the other hand the paper on which this report is based[1] doesn't provide any experimental results of this algorithm being run on a real system – the entire approach may not perform well as there is no guarantee that for instance the pruning criteria can be applied; in this case the technique would end up performing a brute force search on the entire solution space.

A point that is not explored by this algorithm is that of low power operation modes on CPUs. The cost model mentioned above assumes that the power dissipation is constant, but perhaps better performances can be obtained by switching the processor into another power mode during query processing. This of course would add even more solutions to the search space or would impose the need for another approach in exploring the solutions.

Yet another aspect of this algorithm that is worth looking into is that of establishing the value of the parameter *k* – after running the algorithm on several configurations perhaps some heuristics could be established. This can prove valuable as the parameter can for instance change dynamically as the number of requests from other clients to the server reduces, thus allowing the client to work in a more energy efficient mode

7. References.

[1] Rafael Alonso, Sumit Ganguly: Energy efficient query optimization, Technical Report, 1992

[2] Rafael Alonso, Henry F. Korth: Database system issues in nomadic computing, SIGMOD 1993