Energy and Performance Considerations in Work Partitioning for Mobile Spatial Queries

Sudhanva Gurumurthi[†] Ning An[‡] Anand Sivasubramaniam[†]
N. Vijaykrishnan[†] Mahmut Kandemir[†] Mary Jane Irwin[†]

[‡] Oracle Corporation
One Oracle Drive
Nashua, NH 03062, USA
ning.an@oracle.com

[†] Department of Computer Science & Engineering
The Pennsylvania State University
University Park, PA 16802, USA
{gurumurt,anand,vijay,kandemir,mji}@cse.psu.edu

Abstract

A seamless infrastructure for information access and data processing is the backbone for the successful development and deployment of the envisioned ubiquitous/mobile applications of the near future. The development of such an infrastructure is a challenge due to the resource-constrained nature of the mobile devices, in terms of the computational power, storage capacities, wireless connectivity and battery energy. With spatial data and location-aware applications widely recognized as being significant beneficiaries of mobile computing, this paper examines an important topic with respect to spatial query processing from the resource-constrained perspective. Specifically, when faced with the task of answering different location-based queries on spatial data from a mobile device, this paper investigates the benefits of partitioning the work between the resource-constrained mobile device (client) and a resource-rich server, that are connected by a wireless network, for energy and performance savings. This study considers two different scenarios, one where all the spatial data and associated index can fit in client memory and the other where client memory is insufficient. For each of these scenarios, several work partitioning schemes are identified. It is found that work partitioning is a good choice from both energy and performance perspectives in several situations, and these perspectives can have differential effects on the relative benefits of work-partitioning techniques.

1 Introduction

The proliferation of numerous mobile and embedded computing devices has made it imperative to provide users with a seamless infrastructure for information access and data processing. The development of such an infrastructure is a challenge due to the resource-constrained nature of the mobile devices, in terms of the computational power, storage capacities, wireless connectivity and battery energy. With spatial data and location-aware applications widely recognized as being significant beneficiaries of mobile computing, this paper examines an important topic with respect to spatial query processing from the resource-constrained perspective. Specifically, this paper introduces different ways of partitioning the work in the application between a resource-constrained client and a resource-rich server that are connected by a wireless network, and investigates the energy consumption and performance trade-offs between these approaches.

Resource-constraints on a mobile device can bias an application developer to off-load most of the work to a server whenever possible (through a wireless network). For instance, the handheld could serve as a very thin client by immediately directing any user query to a server which may be much more powerful and where energy may not be a problem. The handheld does not need to store any data locally or perform any complicated operations on it in this case. However, it is not clear whether that strategy is the best in terms of the performance and energy viewpoints. Sending the query to the server and receiving the data over the wireless network exercises the network interface, which has been pointed out by other studies to be a significant power consumer [6]. If this dominates over the power consumed by the computing components on the handheld, then it is better to perform the entire operation on the client side as long as the handheld can hold all the data. Sometimes, even if it is energy and/or performance efficient to off-load all the computation to the server, there are several reasons [1] such as access to the server (unreachability/disconnectivity from remote locations) and privacy (the user may not want others, including the server, to be aware of his/her location or queries), causing the entire query execution to be performed on the handheld. There are several options between these extremes - performing all operations at the client or everything at the server - that are worth a closer look. Partitioning the work appropriately between the client and server in a mobile setting can have important ramifications from the performance and energy consumption viewpoints. To our knowledge, no previous study has explored a possibly rich spectrum of work partitioning techniques between the mobile client and the server for mobile spatial database applications to investigate energy-performance trade-offs.

This paper specifically focuses on Spatial Database Management Systems (SDBMS) [17], an important class of applications for the mobile devices. SDBMS are important for mobile computing, with several possible applications in this domain. Already, mobile applications for spatial navigation and querying using a street atlas are available for many PDAs [16, 7]. Queries on spatial data are typically answered in two phases. The first phase, called the *filter-*

ing step, traverses the multidimensional index structure to identify potential candidate solutions. The second phase, called the *refinement* step, performs the spatial operations on each candidate to find exact answers. Refinement can be quite intensive for spatial data (based on the geometric complexity) compared to traditional databases where filtering costs (that is disk intensive) are usually overwhelming. Such well-demarcated phases serve to identify points in the query execution that can be exploited for work partitioning as is explored in this paper. Apart from the two extremes of doing everything at the client or the server, one could envision performing the filtering at the client and the refinement at the server, and vice-versa.

It should be noted that the availability of index structures and data, either partially or completely, on the mobile client has a large influence on the choice of operations done at that end. The filtering step requires the index to be available, and the refinement requires the actual data items from filtering. Shipping a large amount of information back and forth is not a good idea because of the associated communication costs. At the same time, the amount of information that a handheld client can hold is limited by its physical memory size (DRAM) since it does not usually have the luxury of disk storage. Finally, the placement and availability of data is also determined by the frequency of updates (either changes or insertions) since these again involve communication. In our experiments, we consider two scenarios. The first scenario, *Adequate Memory*, assumes that the client has sufficient memory to hold the data set and associated index, while the second scenario, *Insufficient Memory*, assumes that the client can hold only a certain portion of the dataset and index that is limited by its available memory.

This paper explores work partitioning issues for these scenarios using some spatial queries for road-atlas applications on real datasets. A detailed, cycle-accurate performance and energy estimation execution-driven simulator called SimplePower [19], that is available in the public domain, is used to analyze the behavior of these queries. The simulator is augmented with detailed performance and energy [18] models to analyze the wireless communication components. In addition to the software issues of work partitioning and data placement, their interaction with hardware artifacts such as the ratio of the mobile client processor speed to the server processor speed, the wireless communication bandwidth, and the choice of dynamically adjustable power modes is studied.

The rest of this paper is structured as follows. The next section puts this work in perspective with other research endeavors. Section 3 gives a quick background of the spatial queries considered. Section 4 defines the design space of different work partitioning choices that are evaluated. A description of the modeling of the different hardware and software components, including the wireless communication, is given in Section 5, and the results from this evaluation are given in Section 6. Section 7 summarizes the paper.

2 Related Work

The area of mobile computing has drawn a great deal of interest from different research groups. These research efforts address issues at the circuit and architectural level [5], the compiler [13], and also on higher-level issues such as ubiquitous information access [4] and routing protocol design [2].

Several vendors already offer [7, 16] a version of a road atlas (a simple spatial database application). However, many of these applications have been developed in an ad hoc manner, and there is very little prior work on how best to implement such queries on resource-constrained systems. An earlier work [11] looked at the wireless communication and associated energy consumption issues for broadcast data. The problem that this study examined is that of data dissemination: when there is some piece of information that is widely shared, how best to disseminate this data to all mobile devices that may be interested, in a performance/energy efficient manner? The only known prior investigation [1] into query processing for energy/performance efficiency in resource-constrained systems has studied the trade-offs between energy and performance for different spatial index structures. In that study, all the data and index structures were assumed to reside entirely at the client, and no client-server communication was assumed. As was mentioned earlier, it is not clear if this is really the best approach when there is the option of offloading some work to a resource-rich server.

3 Spatial Access Methods and Queries Under Consideration

There has been a great deal of prior work done in the area of storage organizations for spatial (multidimensional) data. Several previous studies have compared these index structures from the performance, scalability, space overheads, simplicity, and concurrency management viewpoints. [1] has looked at some typical spatial index structures from both the performance and energy consumption perspectives for memory resident spatial data. These structures have been pointed out to be representative examples from the design space of storage structures for multidimensional data. Since one of the goals of this work is to investigate how work partitioning compares with performing the entire query at the mobile client, we use the same index structure implementation methodology and query-types here as a reference point.

Line segments (or polylines) dominate road-atlas applications, and these are used as the data items in this study. The queries that have been used in the earlier study, and have been identified as important operations for line segment databases [9, 10] include *point*, *range*, and *nearest-neighbor queries*. In point queries, the user is interested in finding out all line segments that intersect a given point. Range queries are used to select all the line segments that intersect a specified rectangular window. Nearest neighbor queries are proximity queries where the user is interested in finding the nearest line segment from a given point. More details regarding these queries are given in [8]. In the interest of clarity, we uniformly present all evaluation results using the Packed R-tree index structure.

4 Work Partitioning Techniques and Experimental Design Space

There are numerous ways of partitioning the total work (W) between the client and server for the queries. Figure 1 captures the overall structure for work partitioning, where the client performs w_1 amount of work, before sending off a request to the server for it to in turn perform its portion of work (w_2). When the server returns the results from its execution, the client may perform additional work (w_3) before handing the results to the user. It is sometimes possible for the client to overlap its waiting for the results from the server with a certain amount of useful work (w_4) as well. Further, one could also envision communication going back and forth between the client and server, and without significant loss of generality one could capture those scenarios with appropriate values for the w_is.

Where is the computation performed?	Where does the index reside?	Where does the data reside?
Adequate Memory at Client		
Fully at Client	At Client and Server	At Client and Server
Fully at the Server	Only at Server Only at Server	Only at Server At Client and Server
Filtering at Client, Refinement at Server	At Client and Server At Client and Server	Only at Server At Client and Server
Filtering at Server, Refinement at Client	Only at Server	At Client and Server
Insufficient Memory at Client		
Fully at the Server	Only at Server	Only at Server
Fully at the Client	Partly at Client, Fully at Server	Partly at Client, Fully at Server

Table 1. Work Partitioning and Data Placement Choices Explored in This Study.

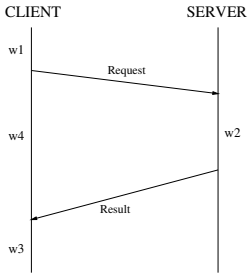


Figure 1. Overall Structure of Work-Partitioning

By associating different values for w_i, one could capture a wide spectrum of work partitioning strategies. Consequently this design space can become exceedingly large, if one were to consider all possible values for the w_is. It should however be noted that work partitioning comes at a cost. One needs to package and transmit/receive state information, and data back and forth, and doing this at a very fine granularity would result in significant overheads. Further, programming can become very hard if one is to consider migrating the work arbitrarily in the middle of an operation (state information has to be explicitly carried over). It is for this reason that we look for explicitly demarcated portions of the code that are at reasonable granularities to consider shipping the work over to the other side. The filtering and refinement steps of query processing offer these clear demarcations in the execution, and we specifically target our work partitioning schemes at the boundaries of these steps. Further, we do not assume any scope for parallelism in the execution, though there could be in reality (this only makes us conservative in our estimate of the benefits of work partitioning), and w_4 is set to zero in our considerations.

Table 1 shows the work partitioning strategies that are actually studied in this paper. The taxonomy is based on where the computation (filtering and refinement) is performed for each query, together with the location of the dataset and associated index. In the first scenario, we examine situations where there is adequate memory at the client. For this we consider,

• Doing everything at the client (i.e. $w_2 = 0$). In this case, the data and index need to be at the client to perform the operations, and this is the only option considered.

• Doing everything at the server (i.e. $w_1 + w_4 + w_3 = 0$). In this case, there is no need to keep the index at the client. While one may avoid keeping the data as well at the client, an advantage with its presence is that the server can simply send a list of object ids after refinement instead of the data items themselves, thus saving several bytes in the results message transmission. Hence, we consider both options for the data at the client.

• Doing filtering at client and refinement at server (i.e. w_1 =filtering, w_2 =refinement). Shipping all the index to the client can incur a high cost. Consequently, we consider only the case where the index is available at the client (and by default, at the server). For the data, as with the previous case, we consider both options to explore the potential of saving communication bandwidth.

• Doing filtering at server and refinement at client (i.e. w_2 =filtering, w_3 =refinement). In this case, it does not make sense keeping the index at the client. Further, since the earlier two cases consider the impact of moving filtered data items from the server, we only consider the situation when the data is already available at the client.

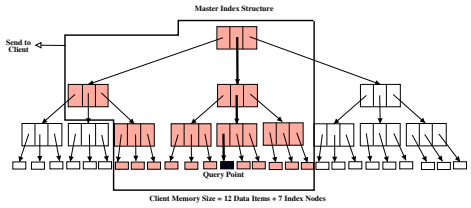


Figure 2. Choosing Data and Index Structure for Shipment to Client

In the second scenario, where the client memory is insufficient, we consider,

• Doing everything at the server (i.e. $w_1 + w_4 + w_3 = 0$). There is no data or index maintained at the client, and all the queries are simply shipped over, and the client just displays the final results it gets back.

• Doing everything at the client. (i.e., w_3 =filtering and refinement). In this case, the client is going to hold data items in a part of the spatial extent and associated index based on its available memory. When a query is given, the client first checks (based on the index it has), whether it can be completely satisfied with its data locally. If so, it goes ahead and does the filtering on the smaller index and the corresponding refinement (i.e. $w_2 = 0$, and there are no messages going back and forth). On the other hand, if it cannot answer the query with local data, it sends the request to the server. The server sends back data items that satisfy the query predicate,

together with some more proximate data items, and an index encompassing all these data items back to the client (w_2 is now the extra work that the server does to pick such data items and build a new index). The amount of data items and index is determined by the client memory availability. The client uses this information to do the filtering and refinement, and stashes this away for future queries. If there is sufficient spatial proximity from one query to the next, then the data transfer costs can be amortized over several queries. We would like to briefly explain how the server picks the data items and new index for the client in this case, since we do not want to incur significant overheads. We use the packed R-tree structure to explain the algorithm. Together with the query, the client sends its memory availability to the server. The server traverses the master index structure (for the entire dataset) as usual except, that it not only picks nodes and data items along the path satisfying the predicate, but also certain nodes on either side of it based on how much data the client can hold. Figure 2 shows an example, where the nodes/data satisfying the actual predicate is shown by the thicker line, while the nodes/data that is actually sent is shown by the shaded area when the client memory size can hold 12 data items and 7 index nodes. Note that we can do this in just one pass down the index structure (as is usual), since the packed R-tree can give reasonable estimates of how many data items and index nodes are present within a given subtree.

One may ask how does the data and/or index get to the client in the schemes which require those to be locally present. If the rewards with such schemes are substantial, then one could advocate either having these placed on the client while connected to a wired network (before going on the road) or incurring a one time cost of downloading this information. We focus on static data/information, and do not consider dynamic updates in this work.

Having described the design space of work partitioning schemes that we consider, we would like to point out that there are several hardware, system software and mobility factors that govern the performance and energy of these schemes. Some of these factors include the relative speeds of the mobile client and resource rich server, the wireless communication bandwidth, the availability of processor power saving modes, the availability of power saving modes on the NIC, and the physical distance between the client and base station. There are also other issues such as noise, packet loss due to mobility, etc. affecting the schemes, and in this study we assume those issues can be subsumed by an appropriate choice of the effective wireless communication bandwidth.

5 Experimental Platform

In the following subsections, we go over the simulation infrastructure which provides a platform for quantifying the performance cycles and energy consumption of the different query execution strategies. We also give details on the workloads (both datasets and queries) and parameters that are used in the evaluations.

5.1 Modeling the Mobile Client

We use the SimplePower [19] simulation infrastructure to model the cycles and dynamic energy consumption of the execution of application code on a 5-stage integer pipeline datapath. This tool is available in the public domain and provides detailed statistics for the different hardware components. The reader is referred to [19] for details on how it works, together with the energy models for the different pieces of hardware - datapath, clock, caches, buses and DRAM memory.

5.2 Modeling the Wireless Communication

In addition to the processor datapath, caches, buses, and memory, we also need to simulate the interface to the wireless NIC, the data communication and the wireless protocol. We have developed a NIC power and timing simulator, together with a protocol simulator, and incorporated this into the SimplePower framework. The effective bandwidth of the channel depends on different parameters such as the channel condition and the underlying modulation/coding schemes used by the client. In this work, we adjust the delivered bandwidth to model the wireless channel condition (errors in wireless transmission).

The NIC model is based on the description presented in [14]. There are four states for the NIC, namely TRANSMIT, RECEIVE, IDLE, and SLEEP, and its power consumption in these states is given in Table 2. The SLEEP state provides the most power saving, but it is physically disconnected from the network. The NIC cannot be used when it is in this mode, and cannot even sense the presence of a message for it (from the server) leave alone receiving it. This state has an exit latency of $470\mu s$ [18] to transition to one of the active modes. The SLEEP state is used before sending the request and after getting back the data to/from the server when we are sure that there will be no incoming message for the client. The IDLE state is used when it is important for the NIC to be able to sense for the presence of a message from the server (when waiting for a response after sending the request to server). The TRANSMIT and RECEIVE states are used when sending and receiving messages respectively. The transmitter is usually much more power consuming than the receiver. This power depends largely on the distance to cover, as is shown in Table 2 for 100 m and 1 Km distances to reach a base station/access point.

State	Power (mW)
TRANSMIT	3089.1 for 1 Km (1089.1 for 100 m)
RECEIVE	165
IDLE	100 (Exit Latency: 0 s)
SLEEP	19.8 (Exit Latency: $470\mu s$)

Table 2. NIC Power States

In addition to the NIC hardware simulation, we have also developed application-level code (APIs) to simulate the software network protocols over the wireless medium. These are executed again on the SimplePower simulator. The APIs include: `SendMessage`, `RecvMessage`, `Sleep`, and `Idle`. The `SendMessage` function simulates the sending of data and `RecvMessage` that of reception (returning when the message is in the appropriate buffers). The API code includes the process of packaging the data into IP packets and TCP segments, and performance/energy costs for this are included in the processor datapath, caches, buses and memory. The `Sleep` and `Idle` functions are used to put the NIC into the SLEEP and IDLE states respectively, and their usage was explained earlier. When the NIC senses an incoming message (when in IDLE mode), it transitions to the RECEIVE state and picks up the message. The duration of its stay in the RECEIVE state is governed by the message length and network bandwidth.

All message transfers include the TCP and IP headers, and are broken down into segments and finally into frames

based on the Maximum Transmission Unit (MTU). The transfer time and energy consumption are calculated based on the wireless bandwidth and the power consumption in the appropriate mode.

In our experiments, in order to minimize the energy consumption of the CPU, when the mobile client is waiting for a message, we let the CPU block itself and make the NIC interrupt it upon a message arrival. Further, when the application is blocked on the client, its CPU is put into a lower power mode.

5.3 Modeling the Server

Since we already have several issues to consider at the client and for wireless communication, which is our main focus in this paper, we make some assumptions at the server end. We assume the server to be resource-rich, in that there are no energy limitations there, and memory is not a problem either (i.e. we assume requests can be satisfied from in-memory index structures and data). Modeling I/O issues and the resulting throughput at the server is part of our future work, and our assumptions here are presuming that there is sufficient locality in the execution (either from the same client or across clients) that the data and associated index nodes get cached in server memory. We believe that relaxing such assumptions would not significantly impact the relative benefits of the work-partitioning schemes that need to go to the server for some information or the other, and we intend to consider such issues in our future work. We also assume that the server is close enough to the wireless access point/ base station and that the costs of getting from the base station to the server are not significant (one could envision relaxing such restrictions in future research as well).

Consequently, all that we need to model at the server end is the performance cycles that are expended in performing its portion of the query after the request has reached a base station. Since we do not need energy simulation, we directly run this code using SimpleScalar [3], a popular superscalar processor simulator, with a different set of parameters [8] than the client (to account for its higher computation capabilities, speed and storage capacities), and feed the resulting performance value back to the (SimplePower + Wireless Network) simulator. This captures the w_2 portion of the execution shown in Figure 1.

5.4 Workload and Simulation Parameters

We have used two line segment datasets from the Tiger database [15]: (a) **PA** contains 139006 streets of four counties - Fulton, Franklin, Bedford and Huntingdon - in southern Pennsylvania, taking about 10.06 MB in size. (b) **NYC** contains 38778 streets of New York City and Union County, New Jersey, taking about 7.09 MB.

The index structure takes around 3.56 MB for the PA dataset and around 1 MB for the NYC dataset. The results in the paper are presented using the PA dataset. The trends for the NYC dataset are similar [8].

For the scenario with adequate client memory, we use the results from 100 runs for each of the three kinds of queries (Point, Range and Nearest Neighbor). Each run uses a different set of query parameters. For the Point queries, we randomly pick one of the end points of line segments in the dataset to compose the query. For the Nearest Neighbor queries, we randomly place the point in the spatial extent in each of the runs. For the Range query, the size (between 0.01% and 1% of the spatial extent), aspect ratio (0.25 to 4) and location of the query windows is chosen randomly from the distribution of the dataset itself (i.e. a denser region is likely to have more query windows). The results presented are the sum total over all 100 runs. The workload generation for the insufficient client memory scenario is discussed later in Section 6.2.

In work partitioning schemes where the required information (data or index nodes) needs to be available on the mobile client, we assume that this information can be downloaded from the server (a one time cost), perhaps even before the user goes on the road with the mobile device.

In our experiments, the server is assumed to have a 4-issue superscalar processor clocked at 1 GHz, with adequate memory to hold all of the dataset and index that are considered. The client is modeled as a single issue processor, with clock speeds that are varied as a fraction of the server clock. The configuration chosen is representative of what is found today in commercial offerings such as the StrongARM SA-1110 [12] (in PocketPCs) which operates at 133 and 206 MHz. Detailed configuration information of the server and the client is given in [8]. We consider two distances - 100 m and 1 Km - for the wireless communication, with communication bandwidths of 2, 4, 6, 8, 11 Mbps.

6 Experimental Results

6.1 Adequate Memory at Client

We first consider the scenario where the client has adequate memory to hold all of the dataset and index if needed. As is pointed out in Table 1, even in this scenario, we consider some situations where the actual data objects are not necessarily present and need to be shipped from the server after a refinement.

6.1.1 Comparison of Schemes Across Queries

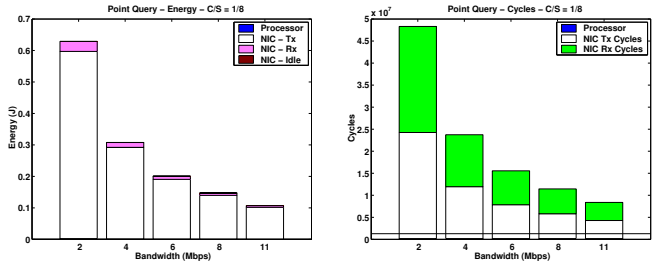
Our first set of results examines the pros and cons of the work partitioning schemes for the three sets of queries that are considered. Figures 3, 4 and 5 compare the schemes for point, range, and nearest neighbor queries respectively with different communication bandwidth parameters for the PA dataset. Since the nearest neighbor query does not have separate filtering and refinement steps, the work partitioning based on these steps are not considered here, and we consider only the options of doing everything at the server versus doing everything at the client. In these experiments, the ratio of the client CPU speed set to server CPU speed (C/S) is set to $\frac{1}{8}$ and the transmission distance is 1 Km.

Results are shown in terms of the (a) energy consumption at the mobile client in the NIC during transmission (NIC-Tx), reception (NIC-Rx) and when idle (NIC-Idle) together with the energy consumption in the other hardware components (datapath, clock, caches, buses, memory) of the client that are clubbed together as Processor (while we have statistics for each component, we do not explicitly show this for clarity); and (b) the total number of cycles from the time the query is submitted till the results are given back (in terms of the time spent by the processor executing its work, and the NIC in transmitting and receiving). In all these graphs, the horizontal solid line (that may appear to be missing in some graphs because of its closeness to the x-axis) represents the corresponding value for the "Fully at the Client" scheme.

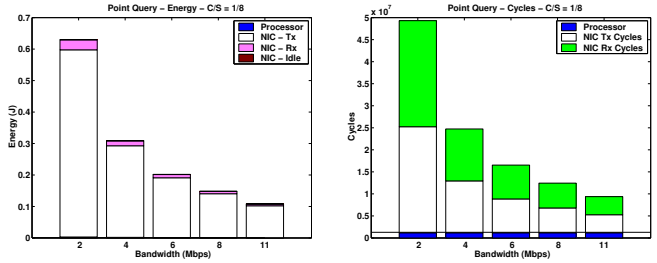
Point Queries: Let us first examine the results for the point queries (Figure 3). This figure compares "Fully at the Client" execution with the energy and performance of

: (i) Fully at Server (Figure 3(a), (ii) Filtering at Client, Refinement at Server (Figure 3(b)) with the index available on the client but the data residing at the server (not available at client), and (iii) Filtering at Server, Refinement at Client (Figure 3(c)) with the index and data available on at the server (after filtering server needs to send actual data items for refinement at client). The reason we do not explicitly show these schemes with data residing at the client option is that this variation does not give very different results. The selectivity of point query is very small, and sending back the data items or just object ids does not alter the resulting message size from the server significantly.

(a) Fully at the Server



(b) Filtering at Client, Refinement at Server



(c) Filtering at Server, Refinement at Client

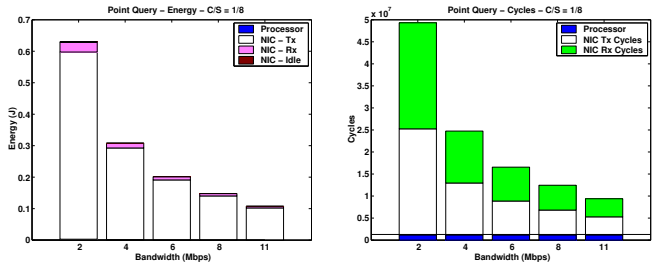
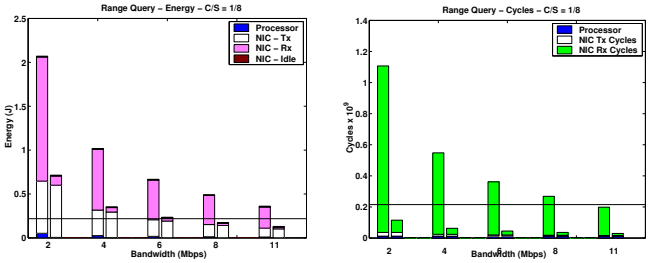


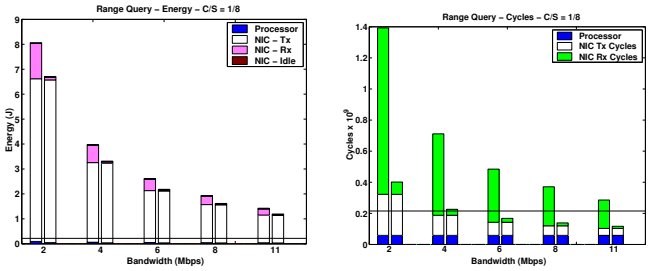
Figure 3. Point Queries. Comparing the schemes in terms of energy consumption on the mobile client, and total cycles taken for the execution. The horizontal line indicates the energy and performance of "Fully at the Client". The profile for energy and cycles is given in terms of what the mobile client incurs in the NIC (given separately for transmission, reception and idle) and all other hardware components that are bunched together as processor.

In all the executions for the point query, we find that both the energy consumption as well as the execution cycles are dominated by the communication portion (especially by the transmitter which has been pointed out to be a big power consumer); processor cycles or energy are not even visible in these graphs. As the transmission bandwidth increases, both energy and cycles drop since the NIC needs to be active for a shorter time and messages do not take as long. Even at 11 Mbps, all these schemes consume much higher energy than doing all the computation locally. The schemes are much worse than full client execution on the performance side as well. Across the three work partitioning schemes that employ the server, we do not find any significant differences between them in terms of the energy or performance behaviors. The reason behind these results is the fact that the point query is not as computationally intense, and the selectivity is much smaller (not only after refinement, but after filtering as well in our experiments). Consequently, the execution (and energy) is dominated by the cost of sending the request to the server rather than by the computation that is performed on either side or the amount of data that is transferred (which usually fits in one packet). One can also observe that the absolute cycles and energy consumed by the query are much smaller than the corresponding values for the range query discussed next.

(a) Fully at the Server



(b) Filtering at Client, Refinement at Server



(c) Filtering at Server, Refinement at Client

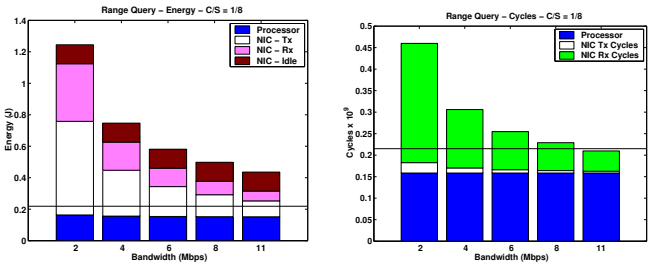


Figure 4. Range Queries. The left bars for (a) and (b) are for the case where data objects are not available at the mobile client and need to be shipped from server, while the right bars are for the case where data objects are already available on the client.

Range Queries: Moving on to the range query in Figure 4, we compare "Fully at the client" execution with the: (i) Fully at the server case (Figure 4(a)), (ii) Filtering at Client,

Refinement at Server case (Figure 4(b)), and (iii) Filtering at Server, Refinement at Client case (Figure 4(c)). For (i) and (ii), the bars on the left for each bandwidth in the corresponding energy and performance graphs show the results for the data residing only on the server i.e., it is not available on the client. Consequently, the server has to send back the data items as well (which take many more bytes than just the object ids) after refinement. The bars on the right in the graphs show the corresponding energy and performance when the data items are available at the client, in which case the server can just send object ids.

There are several interesting observations in the range query results:

• We note that while communication is significant, as in the point query, the processor cycles and energy cannot be discounted in all executions. As the amount of computation that the client performs increases (it increases as we move from (i) to (ii) and to (iii) since refinement is the most time consuming), the processor components of cycles and energy becomes dominating, especially at higher communication bandwidths.

• We find that keeping the data locally helps a lot for (i) and (ii). The benefits are much more apparent for the fully at the server case compared to the other, since the percentage of communication time/energy of the total execution/energy is much higher. We also find that the *benefits of keeping data locally at the client saves much more on performance than on energy*. This optimization only lowers the data reception at the client (from server) and does not alter the transmission of the request to the server. Since the transmitter power is much more dominant, and is unaffected by this optimization, the savings in energy are not as much as the savings in cycles.

• Unlike the point queries, we find that work partitioning does help range queries. With reasonable wireless bandwidths, we can surpass the energy and performance of doing everything at the client in many cases. However, *the performance and energy measures show different points of operating wireless bandwidth at which the work partitioning schemes do better than doing everything at the client*. In general, these schemes start doing better in performance earlier than in terms of energy. This is because the energy costs of communication, are much more expensive than its performance costs, and one needs to go to a much higher bandwidth to offset this difference.

• We also notice differences between the schemes, which we did not find in the point queries. We find the "fully at the server" execution outperforming (ii) and (iii) in terms of both energy and cycles, especially when the data is stored locally at the client. When the data is resident at the client, there is very little communication between the client and server. In fact, this execution outperforms the "fully at the client" execution even at 2 Mbps bandwidth, though it takes over 6 Mbps before it becomes more energy-efficient. Of the other two, we find again a very interesting situation, where the *energy and performance criteria can pick different winners*. Let us examine the cases where the data is available locally on the client. We find that the "filtering at client, refinement at server" is more performance efficient than "filtering at server, refinement at client", and beats the cycles of "fully at client" beyond 4 Mbps. On the other hand, the converse is true in terms of energy. These results can be explained based on the fact that refinement is quite computationally intense, and offloading this to the faster server helps save cycles. However, before doing this the client has to do the filtering and send the candidates from filtering to the server (which is not needed for (iii)). This makes the transmitted message from the client much larger, and as mentioned before, this consumes a lot of power (the energy profiles illustrate this).

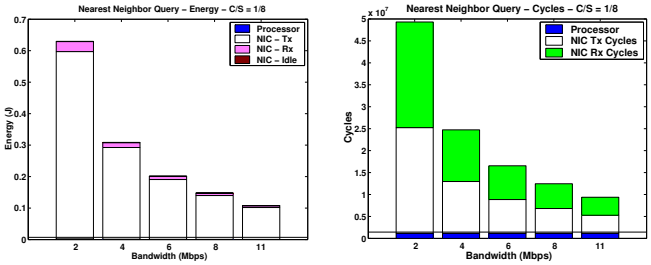


Figure 5. Nearest Neighbor Queries.

Nearest Neighbor Queries: Since we do not have separate filtering and refinement steps in the processing of this query, we have only compared the "fully at server" case with the "fully at client" execution in Figure 5. The selectivity of this query is again quite small, and we have similar observations/results as in the point queries. Here again, it makes sense to do everything at the client as long as we can keep all of the index and data at its memory.

We have also studied the impact of the client CPU speed and the physical distance between the client and base station. Overall, we find that a faster client processor provides greater performance with little impact on the energy consumption, as the overall energy consumption is not significantly affected by the non-NIC components at the client. In terms of the physical distance between the client and the base station, we find that work partitioning schemes that use more transmission power are much more competitive at shorter distances. The detailed results for these experiments can be found in [8].

6.2 Insufficient Memory at Client

In the insufficient memory scenario, we consider the "fully at server" and "fully at client" situations. In the latter scheme, the client directs the first query that it gets to the server. The server examines its index based on the query and ships back a certain amount of data and the corresponding index (as explained earlier) to the client, so that the total data shipped does not exceed x, which indicates client memory availability. Subsequent queries to the client can potentially be satisfied locally if it falls within the spatial extent for the data that it holds. Else, it throws away all the data it has, and re-requests the server for new data and index. The trade-off with this approach is to be able to compensate for the additional costs of transferring the data and index (and the work that the server does in selecting these items) to the client which is absent in the "fully at server" case. However, with sufficient spatial proximity in successive queries (one can expect such proximity in real workloads), this extra cost can be amortized.

To examine this issue, we fire a sequence of queries starting at some random point in the spatial extent, and directing the next set (y) of queries very close to that (so that it can be satisfied locally by the client). We investigate at what values of y (referred to as *spatial proximity*) does the "fully at client" scheme start reaping the benefits for range queries in Figure 6. The investigation examines the effects for $x =$ 1 MB and 2 MB.

We can make a couple of interesting observations. We note that the "fully at client" execution can become energy-efficient beyond a certain number of local queries, compared to sending them all to the server. This number gets higher (from 115 to 200) as we increase the amount of data that is shipped from the server, which reiterates that we need a lot more proximity to offset the higher volume of data

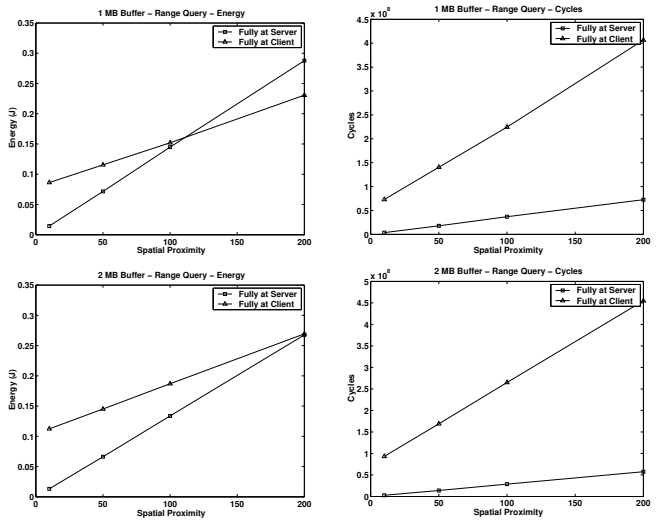


Figure 6. Insufficient Memory at Client - Range Query

transfer. However, the "fully at server" is a clear winner across the spectrum for performance. The client is much slower than the server, and this difference overshadows any wireless transmission cycle overheads that may be incurred otherwise. This is another situation where we find *energy and performance criteria going against each other for optimization*. The energy cost of wireless communication is much more significant than the performance cost, leading to this disparity.

These results suggest that it is important to store a local copy of as small a set of spatially proximate data items that the user may need to save on both energy as well as performance.

7 Summary of Results and Future Work

Energy constraints and performance goals are two important criteria that one needs to keep in mind when designing applications for mobile systems. A previous study [1] looked at this issue for the first time by examining spatial data indexing - that can benefit several mobile/location-aware applications - from the energy and performance angles. However, there was a limitation in that study since it was assumed that all the data is available on the mobile client and all operations are directly performed on it. In this paper, we have addressed this limitation by examining different ways of implementing/partitioning spatial queries between the mobile client and server, and examining energy and performance benefits, if any, from these approaches. By identifying a set of issues and strategies that need to be investigated in partitioning the work between a client and server across a wireless network, we hope to provide a more systematic way of designing and implementing applications for this environment in a performance and energy efficient manner. This effort is intended to be the first step towards this goal, and there are several issues that warrant further investigation, such as exploiting parallelism and pipelining and consideration of other spatial queries.

Acknowledgements

This research has been supported in part by NSF grants: 0103583, 0097998, 9988164, 0130143, 0093082, and 0103583, 0082064, NSF CAREER Awards 0093082 and 0093085, and MARCO 98-DF-600 GSRC.

References

[1] N. An, A. Sivasubramaniam, N. Vijaykrishnan, M. Kandemir, M. Irwin, and S. Gurumurthi. Analyzing Energy Behavior of Spatial Access Methods for Memory-Resident Data. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 411–420, September 2001.

[2] H. Balakrishnan. *Challenges to Reliable Data Transport over Heterogeneous Wireless Networks*. PhD thesis, University of California, Berkeley, August 1998.

[3] D. Burger and T. Austin. The SimpleScalar Toolset, Version 2.0. Technical Report 1342, University of Wisconsin, June 1997.

[4] R. Cerqueira, C. Hess, M. Román, and R. Campbell. Gaia: A Development Infrastructure for Active Spaces. In *Proceedings of the Workshop on Application Models and Programming Tools for Ubiquitous Computing (held in conjunction with the UBICOMP 2001)*, September 2001.

[5] A. Chandrakasan and R. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[6] P. Gauthier, D. Harada, and M. Stemm. Reducing Power Consumption for the Next Generation of PDAs: It's in the Network Interface. In *Proceedings of the International Workshop on Mobile Multimedia Communications (MoMuC)*, September 1996.

[7] GEOPlace.Com. Mobile Technology Takes GIS to the Field. http://www.geoplace.com/gw/2000/0600/0600IND.ASP.

[8] S. Gurumurthi, N. An, A. Sivasubramaniam, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Energy and Performance Considerations in Work Partitioning for Mobile Spatial Queries. Technical Report CSE-01-028, Dept. of Computer Science and Engineering, The Pennsylvania State University, November 2001.

[9] E. G. Hoel and H. Samet. Efficient Processing of Spatial Queries in Line Segment Databases. In *Proceedings of the 2nd Symposium on Advances in Spatial Databases(SSD)*, pages 237–256, Zurich, Switzerland, August 1991. Lecture Notes in Computer Science, Vol.525, Springer.

[10] E. G. Hoel and H. Samet. A Qualitative Comparison Study of Data Structures for Large Line Segment Databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 205–214, San Diego, California, June 1992. ACM PRESS.

[11] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy Efficient Indexing on Air. In *Proceedings of the ACM Conference on Management of Data (SIGMOD)*, pages 25–36, 1994.

[12] Intel StrongARM SA-1110 Microprocessor Brief Datasheet. http://developer.intel.com/design/strong/datashts/278241.htm.

[13] U. Kremer, J. Hicks, and J. Rehg. A Compilation Framework for Power and Energy Management in Mobile Computers. In *Proceedings of the 14th International Workshop on Parallel Computing (LCPC 2001)*, August 2001.

[14] LMX3162 Single Chip Radio Transceiver. National Semiconductor Corporation, March 2000. http://www.national.com/pf/LM/LMX3162.html.

[15] R. W. Marx. The TIGER System: Automating the Geographic Structure of the United States Census. *Government Publications Review*, 13:181–201, 1986.

[16] Microsoft. Microsoft Pocket Streets. http://www.microsoft.com/mobile/downloads/streets.asp.

[17] S. Shekhar, S. Chawla, S. Ravada, et al. Spatial Databases - Accomplishments and Research Needs. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):45–55, 1999.

[18] E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks. In *Proceedings of the ACM SIGMOBILE Conference of Mobile Computing and Networking (MOBICOM 2001)*, July 2001.

[19] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. Kim, and W. Ye. An energy estimation framework with integrated hardware-software optimizations. In *Proceedings of the International Symposium on Computer Architecture*, 2000.