# Energy and Performance Considerations in Work Partitioning for Mobile Spatial Queries

Sudhanva Gurumurthi
Ning An
Anand Sivasubramaniam
N. Vijaykrishnan
Mahmut Kandemir
Mary Jane Irwin

Technical Report CSE-01-028
November, 2001

Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802.
E-Mail: anand@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. This infrastructure should allow a user to access and process information from anywhere, whether it be from a desktop or on the road. Further, the user should not be able to perceive any noticeable differences in the performance across these different environments. These goals become particularly challenging when constrained by the resource availability on many 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. The execution of spatial queries using a packed R-tree index structure is modeled for each of these schemes using a cycle accurate performance and energy simulator, that captures the wireless communication as well. Three different spatial queries on different datasets are studied for these schemes by varying several hardware and system software parameters. The results show that the type and nature of queries has an important influence on the choice of work partitioning schemes, as does the dataset, relative speed of the mobile client and server CPUs, and the wireless network bandwidth. 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.

**Keywords:** Spatial Data, Energy Optimization, Multidimensional Indexing, Resource-constrained and Mobile Computing.

## 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. This infrastructure should be seamless from two perspectives. First, one should be able to access and process information from anywhere - whether sitting at a desktop or from a mobile device on the road - in a transparent manner, without any perceptible differences between these environments. The computational capabilities, storage capacities, network connectivity, mobility or user interface differences across these diverse environments should not pose any restrictions on the kinds and difficulty of applications that the user may want to run. Second, one should not notice any perceptible differences in the performance (responsiveness and throughput) across these environments as well. These two perspectives are key requirements for the accelerated deployment and success of the envisioned ubiquitous computing environment. The resource-constraints in many of the mobile devices pose several challenges in making them responsive to the increasing demands of applications that are being imposed by users. This paper explores issues from these two perspectives in running spatial database applications - an important class of mobile applications - on handheld devices, examining the impact of resource constraints (battery energy in particular) on performance in a wireless environment. 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.

The mobile computing environment poses several unique problems, such as, limited computational resources, battery energy, and network connectivity, of which the battery energy is considered to be the most challenging to tackle. The growing mismatch between energy capacity of batteries and the energy consumption of mobile devices makes it critical to employ algorithmic, software and architectural techniques for energy savings, in addition to traditional circuit and gate-level optimizations. It is hypothesized that high level optimizations in algorithms and data structures can give much more energy savings than micro-managing the energy consuming resources at a very low level.

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 [9]. 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 [2] 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.

Database applications are expected to be the dominant workloads running on the mobile devices [1]. Apart from the prevalent personal organizer database applications (address book, calendar, etc.), there are numerous productivity-enhancing commercial, entertainment and convenience-based database applications envisioned for such devices. This paper specifically focuses on spatial databases, an important class of applications for the mobile devices. In general, Spatial Database Management Systems (SDBMS) [28] have found widespread adoption in numerous areas including Geographical Information Systems (GIS), Image Processing, Military Planning and Logistics, Computer Aided Design (CAD), Multimedia Systems, and Medical Database Systems. 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 [21, 10]. In addition, traditional data input and querying for conventional SDBMS can be supplanted by mobile operations for better productivity and convenience.

The motivation for database management in mobile/pervasive environments has been very elegantly pointed out in a recent tutorial at the VLDB 2001 conference [8], together with a long list of ongoing and future important research topics in this area. This tutorial stresses the importance of data management on small form factor devices that are limited in memory, battery energy and wireless connectivity. Spatial data management and location-aware operations are identified as key target applications with several examples.

SDBMS design and implementation is a difficult problem [28] even on conventional systems because it has to deal with multidimensional data. Data objects have varying sizes associated with them, and spatial operations are in general much more complex than standard relational operators. Further, the efficiency of the storage organization is highly dependent on the nature and idiosyncrasies of the spatial dataset. Moving the target to a mobile device makes the design and implementation of a SDBMS even more challenging. Resource constraints such as limited energy, computational power and memory add to the complexity of the problem. Performance is not necessarily the only goal for optimization. Sometimes the user may be willing to sacrifice some amount of performance if that will enable the device to run longer on battery. Power dissipation of different system components may also be an important issue for thermal considerations.

Queries on spatial data are typically answered in two phases. The first phase, called the *filtering* 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.

With the goal of analyzing energy-performance trade-offs when partitioning the work involved in spatial data queries between a resource-rich server and a resource-constrained handheld device (client), this paper investigates the following issues:

- *Adequate Memory:* We explore four options - (a) performing the query fully at the client, (b) performing the filtering at the client and the refinement at the server, (c) performing the filtering at the server and the refinement at the client, and (d) performing the query fully at the server - for different spatial queries. It should be noted that of these options, (a) demands the maximum amount of client memory, while the other three demand less. We also consider two variations of these options (for (b), (c) and (d)) where the actual data can be absent or present on the client. Absence of the data on the client implies that the server has to send the actual data items (which can take up many more bytes) to client, while it can simply send object ids when the data is present on the client.

- *Insufficient Memory:* When the data set and associated index is much larger than the available client memory, we investigate the following two options for query execution: (a) transferring a part of the data (and the index for that portion of the data) that is spatially close to the query region over to the client based on its available memory, and then performing the query at the client end, and (b) performing the query fully at the server. The former option can be a good idea if there is sufficient spatial locality from one query to the next, which is quite possible in a real workload. If the next set of queries is going to be focussed on a spatially proximate region of the first query, then they could perhaps be satisfied by the mobile client itself. This could amortize the cost of the larger data transfers for the first query. Trade-offs between spatial proximity in the workload and size of the data that is transfered are evaluated for this option.

This paper explores these work partitioning issues using some spatial queries for road-atlas applications on real datasets. A detailed cycle-accurate performance and energy estimation execution-driven called SimplePower [30], that is available in the public domain, is used to analyze the behavior of these queries. The evaluation framework also uses detailed performance and energy models to analyze the wireless communication components [29]. 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 spatial index structures (specifically the packed R-tree [17] that is used here), and the implementation of the spatial queries considered here on such structures. 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 lessons learned from this exercise, and identifies directions for future research.

## 2 Related Work

The area of mobile computing has drawn a great deal of interest from different research groups. Hardware engineers and system architects have tried to provide small form factor components that are energy-efficient by attempting gate, circuit, and architectural-level innovations [7]. Compilers and runtime systems have developed techniques for working in resource-constrained settings [18]. In terms of the systems software, there have been research projects to integrate information from different sources, and to provide ubiquitous access to this information [6]. Wireless technology and routing protocols for mobile devices have also advanced significantly to make anytime-anywhere connectivity a reality [3].

All these innovations have started providing the infrastructure for building many mobile applications. While the most popular commercial ones are personal organizers (small databases), productivity enhancing tools and games, the more relevant applications related to this work are those that are found in the spatial database and GIS area. Several vendors already offer [10, 21] a version of a road atlas (a simple spatial database application), allowing the user to get driving directions (shortest path problem), examine detailed map information (range queries), give details on a landmark/restaurant that the user points to (point queries), and so on. 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. The only known prior investigation [2] 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 this study, all the data and index structures are assumed to reside entirely at the client, and no client-server communication or work partitioning is considered. 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 over a wireless network.

An earlier work [15] 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? This is an important problem for situations when several mobile devices are interested in the same information (and the amount of information to be disseminated is not too large), where broadcast becomes an effective solution. In this work, however, we are looking at a more general problem where different users are possibly interested in different pieces of information, and this information can be much larger. So it is essential for the client to initiate and transmit the queries to the server, and for the server to respond individually to each client.

## 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. As mentioned in the previous section, a recent paper [2] has looked at three spatial index structures - PMR QuadTrees [22, 13, 14], Packed R-Trees [17], and Buddy Trees [26] - 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 structures and queries here as a reference point. The following discussion gives a brief overview of these queries and index structures.

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 [13, 14], include:

- **Point Queries:** In these queries, the user is interested in finding out all line segments that intersect a given point. For instance, such an operation could be used to find out the name of a street that is pointed to by the user, or to find out which streets meet at a given intersection.

- **Range Queries:** These queries, also known as spatial selection or window queries, are used to select all line segments intersecting with a specified rectangular window. Very often, the user wants to magnify a portion of the atlas for a closer examination, and this query can serve such a request.

- **Nearest Neighbor Queries:** These are proximity queries where the user is interested in finding the nearest line segment (street) from a given point (e.g. what is the closest street to a given landmark, subway station, etc.). This is the perpendicular distance to the line segment if the perpendicular intersects the segment, and is the distance to one of the end points (closest one) otherwise.

Range and Point queries are typically implemented using a *filtering step* where the possible candidates are first identified using their minimum bounding rectangles (MBRs). Each index node of the hierarchical spatial structures represents a rectangular region of the spatial extent that it covers, and is represented by the MBR of this region. The filtering step, that traverses the index structure, uses these MBRs to identify possible candidates. Subsequently, a *refinement step* is needed to perform the actual geometric operations on each short-listed data item to find the exact answers to the query.

The Nearest Neighbor query is a little more complicated to implement with different previous suggestions [13, 23, 24]. For instance, [23, 24] uses a progressively expanding (in size) range query centered around the query point till the first data item is found. Another possibility [13] is to actually go to that region of the index structure, and examine around this region in the structure instead of composing the searches as separate range queries. A more interesting, and perhaps more efficient, approach is studied in [24] and is the strategy used in this paper. The search starts at the root node and examines the MBRs of its children. It orders these MBRs in terms of distances from the query point, and uses these distances to determine the recursive search order. In addition, it also uses these distances to prune the search when noticing that certain MBRs will definitely contain data items that are closer than those for the other children. The process is then recursively carried out for the candidate child nodes. This is a general technique that can be used for any of the considered hierarchical spatial access methods. The nearest neighbor query does not have separate filtering and refinement steps in our implementation.

In the interest of clarity, this paper uniformly presents all evaluation results using the Packed R-tree structure. The R-tree [12] was initially proposed as an extension to the B-Tree structure for handling multidimensional data. Many variants of the R-Tree, such as $R^+$-Tree [27] and $R^*$-Tree [4] have been studied. They differ in the algorithm that is used for insertion, specifically in splitting a node of the tree when its subtree is filled. They attempt to give better balanced (and efficient) trees by dynamically adapting to the insertion pattern/sequence. However, these structures can become inefficient when the database of spatial items is static (and known *a priori*). In such cases, one should use bulk-loading techniques rather than insert item by item to build the data structure. Roussopoulos and Leifker [25] use packed R-trees for such

| Where is the computation performed? | Where does the index reside? | Where does the data reside? |
|---|---|---|
| | | |
| **Adequate Memory at Client** | | |
| Fully at the Client | At both Client and Server | At both Client and Server |
| Fully at the Server | Only at the Server<br>Only at the Server | Only at the Server<br>At both Client and Server |
| Filtering at Client, Refinement at Server | At both Client and Server<br>At both Client and Server | Only at the Server<br>At both Client and Server |
| Filtering at Server, Refinement at Client | Only at the Server | At both Client and Server |
| | | |
| **Insufficient Memory at Client** | | |
| Fully at the Server | Only at the Server | Only at the 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.

static databases to lower response times. Further, Kamel and Faloutsos [17] suggest using Hilbert value (a linearization technique for multidimensional space [11]) for sorting the data items before constructing the bulk-loaded R-tree. This is the structure that is used in this paper. Typically, such R-trees are built in a bottom-up fashion, level by level. After the line segments are sorted, for each line segment, starting from the first and going one after another, a pointer and its MBR are entered into an index node. When the number of pointers exceeds the node capacity, a new index node is created. After all the lines are assigned pointers and MBRs, index nodes for the next higher level are created to point to the index nodes at the lower level, and the process continues recursively till we get a single root index node. Along with each pointer, we keep track of the MBR of the area that is covered by that subtree. The R-tree structure allows MBRs of pointers to overlap, which actually helps keep it balanced. However, the downside to this is that a search has to traverse more possible paths in the hierarchical structure. Depth-first traversal is used to implement the queries.

## 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 described in the previous section. 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 $w_i$s.

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 $w_i$s. 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

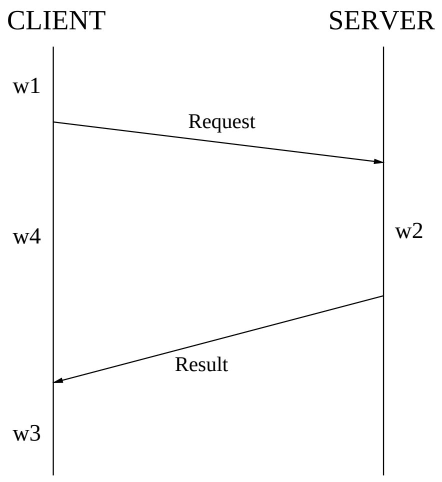


Figure 1: Overall Structure of Work-Partitioning

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.

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.

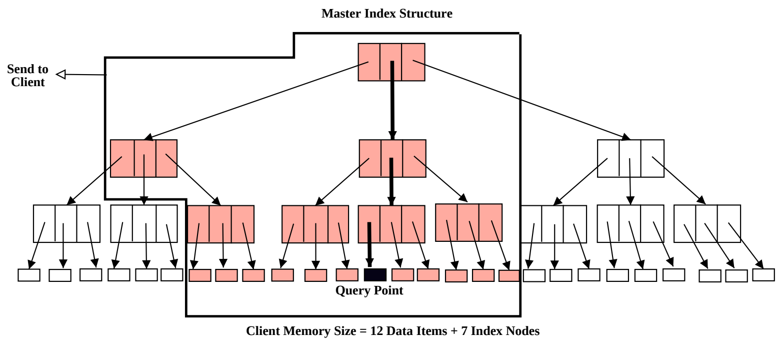


Figure 2: Choosing Data and Index Structure for Shipment to Client

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 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. As was mentioned earlier, 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. Consequently, their implementation warrants close scrutiny. Some of these factors include:

- *The relative speeds of the mobile client and resource rich server*: A very slow client can be biased towards the scheme which offloads all the work to the server, while a very fast client can bias the other way around. In our experiments, we vary the ratio of these speeds to study its effect.

- *Wireless Communication Bandwidth*: This not only has an impact on the execution time for schemes sending messages back and forth, but also has a bearing on energy consumption since the wireless Network Interface Card (NIC) has to remain active longer during transmission/reception.

- *Processor Power Saving Modes*: Several mobile (low power) CPUs offer power saving modes to which the software can transition the processor during periods of inactivity. This would be particularly useful for schemes incurring long wait times before they can receive data from the server.

- *Wireless Interface Power Saving Modes*: Just as the CPU provides power saving modes, the wireless interface also has different power modes that can be used during periods of inactivity. There are trade-offs between transitioning costs between these modes and power savings can you can get.

- *Physical Distance Between Client and Base Station*: An important consideration in the power consumption of the wireless transmitter (not the receiver) on the handheld device is the physical distance that the signals need to traverse before reaching the base station/access point. For instance, changing the transmission distance from 100 meters to 1 kilometer can nearly triple the transmitter power. Consequently, locality preserving schemes are expected to do much better with such increasing distances.

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.

## 4.1 Quantifying Trade-offs when Partitioning the Work

Consider the following parameters that can capture the behavior of different work partitioning strategies:

- $B$ - Effective (delivered) wireless bandwidth
- $E_{fully\_local}$ - Energy consumption of the computation if performed completely on the client (in Joules) (This is the energy consumed when $w_2 = 0$)
- $E_{local}$ - Energy consumption of that portion of the overall computation that is performed locally. (This is the energy consumed during portions $w_1$ and $w_3$)
- $E_{protocol}$ - Energy consumption by the processor on the client in network protocol processing (does not include wireless interface energy).
- $C_{fully-local}$ - Client processor cycles spent executing the entire computation locally.
- $C_{protocol}$ - Client processor cycles spent in network protocol processing.
- $C_{local}$ - Client processor cycles spent in that portion of the overall computation that is performed locally.
- $C_{Tx}$ - Number of client cycles for transmitting the data from the wireless interface.
- $C_{wait}$ - Number of client cycles waiting before reception is initiated.
- $C_{Rx}$ - Number of client cycles spent receiving the data at the wireless interface.
- $C_{w_2}$ - Number of server cycles processing the query at server side.
- $Mhz_C$ - Clock speed of the client
- $Mhz_S$ - Clock speed of the server
- $P_{Client}$ - Power consumption (in Watts) by the client in the datapath, caches, buses, and memory (does not include the wireless interface)
- $P_{Tx}$ - Wireless NIC power consumption during transmission
- $P_{Rx}$ - Wireless NIC power consumption during reception
- $P_{idle}$ - Wireless NIC power consumption when idle
- $P_{sleep}$ - Wireless NIC power consumption when put to sleep (consumes lesser power than the idle mode)
- $Packet_{Tx}$ - Transmission packet-size (in bits)
- $Packet_{Rx}$ - Reception packet-size (in bits)

Then, the number of cycles for transmitting a packet of size $Packet_{Tx}$ is given by

$$C_{Tx} = (\frac{Packet_{Tx}}{B})Mhz_C$$

Similarly, the time for receiving a packet of size $Packet_{Rx}$ is

$$C_{Rx} = (\frac{Packet_{Rx}}{B})Mhz_C$$

The number of client cycles spent while the server is performing the work is given by

$$C_{wait} = (\frac{C_{w_2}}{Mhz_S})Mhz_C$$

Further, let us define $E_{fully-local}$ and $E_{local}$ as

$$E_{fully-local} = (P_{client} + P_{sleep})C_{fully-local}$$

Therefore, in order for work partitioning to be effective: From the performance perspective, we need to have

$$C_{fully-local} > C_{Tx} + C_{w_2} + C_{Rx} + C_{local} + C_{protocol}$$

From the energy perspective, we need to have

$$(P_{client} + P_{sleep})C_{fully-local} > Mhz_C[P_{Tx}(\frac{Packet_{Tx}}{B}) + P_{Rx}(\frac{Packet_{Rx}}{B}) + (P_{idle} + P_{client})(\frac{C_{w_2}}{Mhz_S} + C_{local} + C_{protocol})]$$

Intuitively, it may appear from the above relations, that one could reduce energy consumption by one of more of the following:

- Increasing the effective network bandwidth ($B$)
- Reducing the amount of data transferred ($Packet_{Tx}$ and $Packet_{Rx}$)
- Reducing the ratio of the client processor speed to that of the server ($\frac{Mhz_C}{Mhz_S}$)
- Reducing the server-computation time ($C_{w_2}$)
- Reducing the power-consumption of the processor and the wireless interface ($P_{client}, P_{Tx}, P_{Rx}, P_{idle}$, and $P_{sleep}$).
- Reducing the energy consumption of local client computation ($E_{local}$)
- Reducing the energy consumption of the protocol implementation ($E_{protocol}$).

There are complex interactions between the amount of data that needs to be transferred and the computations that are performed at the client. There are interactions between the nature of the computation and the size of the result with factors such as the bandwidth, speed of the client and server, and the protocol overhead. A detailed empirical evaluation of the different choices is needed for a clear understanding of this design space. Our experimental platform uses a simulation infrastructure to conduct this empirical evaluation.

## 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 [30] 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 [30] 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 Network

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 [19]. 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 $470\mu s$ [29] 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 ($B$) and the power consumption in the appropriate mode.

In addition to the NIC power state choices, there are also several design choices for the code running on the mobile device in managing the CPU during communication. For instance, what should the CPU do when waiting for a message? Should it continue to poll the message-queue state variables in a busy-wait loop, or should it block itself and let the NIC interrupt it upon message arrival? The advantage with busy waiting is that there is little time lost between message arrival and is thus effective from the performance viewpoint. However, this implementation not only exercises the processor datapath components while executing the loop, but also keeps hitting the L1 caches (particularly the I-cache) which has been shown to be a significant energy drainer from the overall system perspective [2]. On the average, we found that the latter approach of blocking the CPU during a receive operation cut the energy consumption in this operation by more than half, and consequently present results only using the blocking option in the performance results.

Further, when the application is blocked on the client, the client CPU can possibly be put into a lower power mode for the CPU (many mobile versions of processors offer multiple power modes including the mobile Pentium, Transmeta, and StrongARM). Upon an interrupt from the NIC, it transitions out of this state and becomes active again to pick up the message. In our experiments we find that this option gives a saving between 10-20% of energy savings in several cases, and use this mode whenever the client is blocked during communication.

## 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 [5], a popular superscalar processor simulator, with a different set of parameters 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

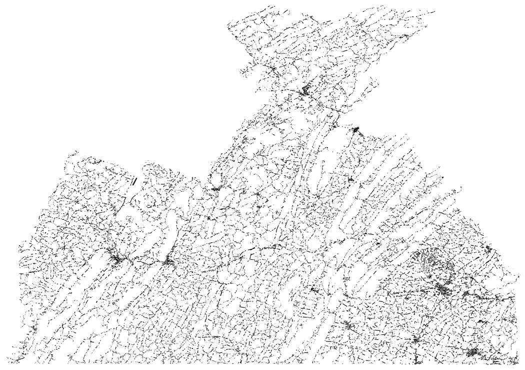
We have used two line segment datasets from the Tiger database [20]: (a) **PA** contains 139006 streets of four counties - Fulton, Franklin, Bedford and Huntingdon - in southern Pennsylvania, taking about 10.06 MB in size. (b) **NYC** contains 38778 streets of New York City and Union County, New Jersey, taking about 7.09 MB. A pictorial view of these datasets is given in Figure 3.

This structure takes around 3.56 MB for the PA dataset and around 1 MB for the NYC dataset. Most of the results are presented using the PA dataset, and we show one set of results with the other dataset to show that the trends are similar.

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 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.

Tables 3 and 4 list the simulation parameters chosen 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.


(a) PA - Southern Pennsylvania Counties

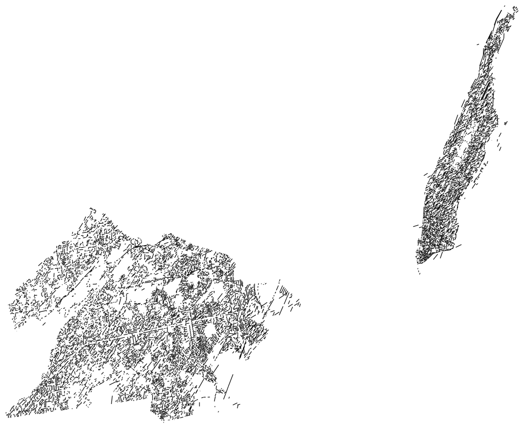

(b) NYC - New York City and Union County, NJ

Figure 3: Datasets

| Parameter | Value |
|---|---|
| Processor Clock Speed ($Mhz_C$) | $Mhz_S/8$, $Mhz_S/4$, $Mhz_S/2$, $Mhz_S/1$ |
| Processor Organization | Single-Issue 5-stage pipelined integer datapath |
| I-Cache Configuration | 16 KB 4-way set-associative |
| D-Cache Configuration | 8 KB 4-way set-associative |
| I- and D-Cache Line Size | 32 bytes |
| Cache Hit Latency | 1 cycle |
| Memory Size | 32 MB |
| Memory Access Latency | 100 cycles |
| Supply Voltage | 3.3 V |
| Feature Size | 0.35 micron |

Table 3: Client Configuration

| Parameter | Value |
|---|---|
| ILP | 4 |
| RUU Size | 64 |
| Load/Store Queue Size | 32 |
| Functional Units | 2 Ints,2 FP |
| Memory Size | 128 MB |
| Cache Hierarchy | Two-Level with L1 D-Cache and I-Cache and Unified L2 Cache |
| Instruction Cache Size | 32KB |
| Instruction Cache Line Size | 64B |
| Instruction Cache Associativity | 2 |
| Data Cache Size | 32 KB |
| Data Cache Line Size | 64B |
| Data Cache Associativity | 2 |
| L2 Cache Size | 1MB |
| L2 Cache Line Size | 128B |
| L2 Cache Associativity | 2 |
| Unified TLB (fully assoc) entries | 64 |
| Clock Speed | 1 GHz |

Table 4: Server Configuration

The client is modeled as a single issue processor, with clock speeds that are varied as a fraction of the server clock. These values are representative of what is found today in commercial offerings such as the StrongARM SA-1110 [16] (in PocketPCs) which operates at 133 and 206 MHz. We consider two distances - 100 m and 1 Km - for the wireless communication, with communication bandwidths of 2, 4, 6, 8, 11 Mbps (which is in the range of what is available or expected to be soon available in commercial offerings).

## 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

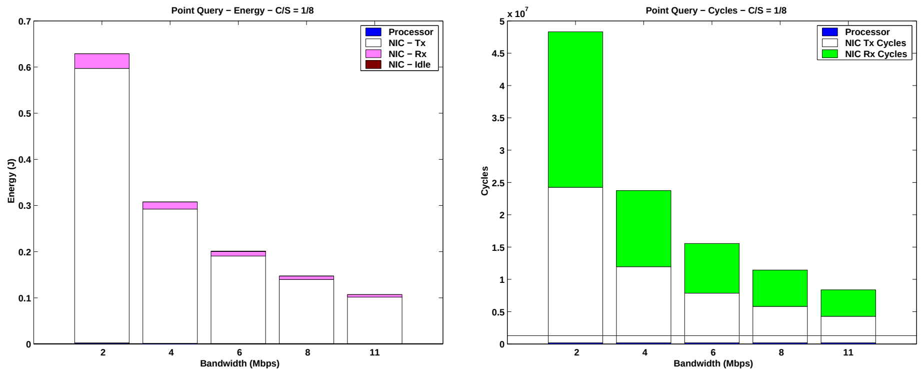
Our first set of results examines the pros and cons of the work partitioning schemes for the three sets of queries that are considered. Figures 4, 5 and 6 compare the schemes for point, range, and nearest neighbor queries respectively with different communication bandwidth parameters for the PA dataset. 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 client CPU speed ($Mhz_C$) is set to $Mhz_S/8$ 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.

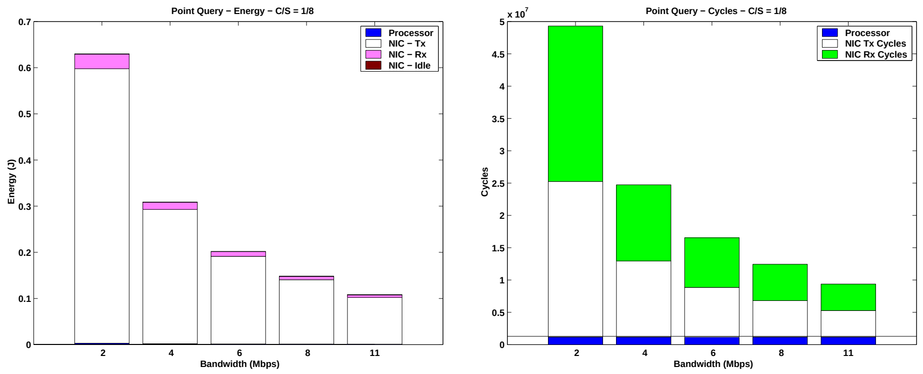
**Point Queries:** Let us first examine the results for the point queries (Figure 4). This figure compares ”Fully at the Client” execution with the energy and performance of : (i) Fully at Server (Figure 4(a), (ii) Filtering at Client, Refinement at Server (Figure 4(b)) with the index available on the client but the data residing at the server (not available at client), and (ii) Filtering at Server, Refinement at Client (Figure 4(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.

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

(a) Fully at the Server



(b) Filtering at Client, Refinement at Server



(c) Filtering at Server, Refinement at Client

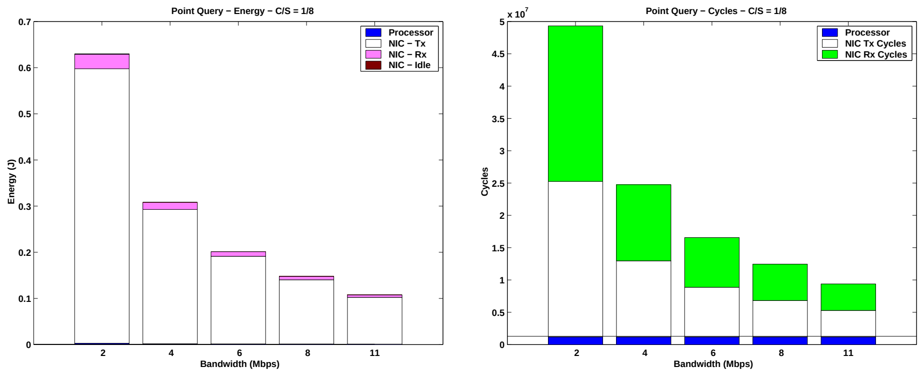
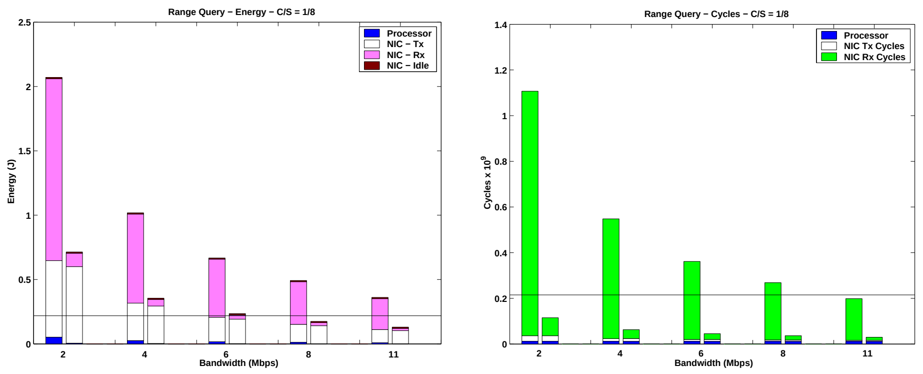
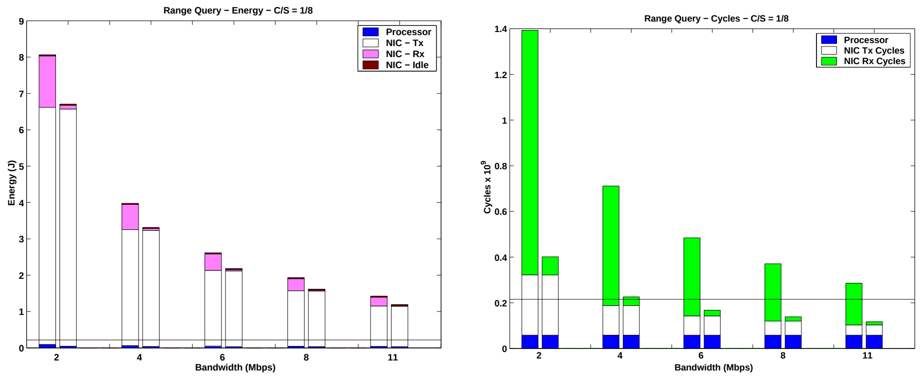


Figure 4: Point Queries. Comparing the schemes in terms of energy consumption on the mobile client, and total cycles taken for the execution. The horizontal line indicates the energy and performance of ”Fully at the Client”. The profile for energy and cycles is given in terms of what the mobile client incurs in the NIC (given separately for transmission, reception and idle) and all other hardware components that are bunched together as processor.

(a) Fully at the Server



(b) Filtering at Client, Refinement at Server



(c) Filtering at Server, Refinement at Client

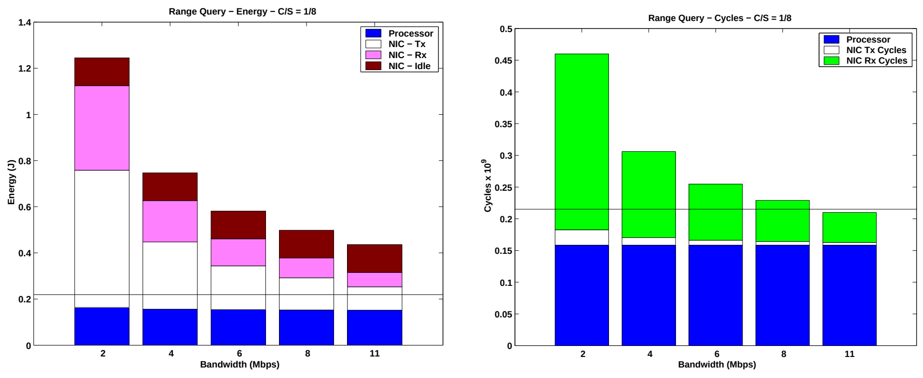


Figure 5: Range Queries. Comparing the schemes in terms of energy consumption on the mobile client, and total cycles taken for the execution. The horizontal line indicates the energy and performance of ”Fully at the Client”. The profile for energy and cycles is given in terms of what the mobile client incurs in the NIC (given separately for transmission, reception and idle) and all other hardware components that are bunched together as processor. 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.

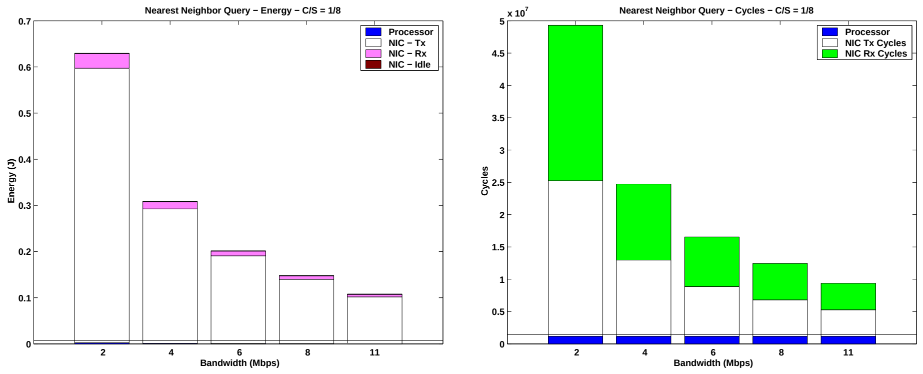


Figure 6: Nearest Neighbor Queries. Comparing ”Fully at the Server” execution with ”Fully at the Client” that is shown as the horizontal line. The profile for energy and cycles is given in terms of what the mobile client incurs in the NIC (given separately for transmission, reception and idle) and all other hardware components that are bunched together as processor.

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.

**Range Queries:** Moving on to the range query in Figure 5, we compare ”Fully at the client” execution with the: (i) Fully at the server case (Figure 5(a)), (ii) Filtering at Client, Refinement at Server case (Figure 5(b)), and (iii) Filtering at Server, Refinement at Client case (Figure 5(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 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).

**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 6. 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.

### 6.1.2 Sensitivity to Dataset

Previous results were shown for the PA dataset, and we now examine the differences when we move to the NYC dataset using the range queries (Figure 7). The difference between the datasets is that NYC is a little smaller. More importantly, the selectivity from the filtering step in NYC is smaller than the corresponding selectivity in PA. One can observe that the transmission (where the ids from filtering need to be moved to server) energy or cycles in the ”Filtering at Client, Refinement at Server” for NYC is lower than those for PA. Similarly, the receive energy or cycles in ”Filtering at Server, Refinement at Client” is lower for NYC. This factor makes these hybrid schemes much more competitive than the corresponding executions for PA.

### 6.1.3 Impact of Client CPU Speed

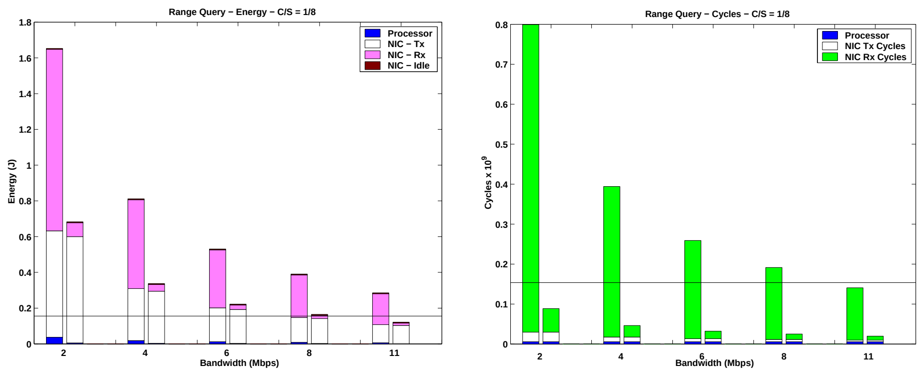
The clock speed for the mobile client was set to one-eighth the server speed in earlier experiments, that is reflective of many current commercial offerings. In order to investigate what effect the relative speed between the client and server has on the execution, we show in Figure 8 the range query results with the client speed set to half that of the server. With a faster client, we clearly see that the ”Fully at Client” execution takes lower time to complete (the cycles in this graph refer to the new client clock which has much higher frequency). So performance-wise, the faster client definitely helps the schemes which do more work at the client (”Fully at Client”, followed by ”Filtering at Server, Refinement at Client”, and then ”Filtering at Client, Refinement at Server”).

On the energy angle, note that the overall energy is not significantly affected by whatever is consumed by the non-NIC components at the client, including the processor. Since the NIC usage time is determined primarily by the bandwidth rather than by the speed of the client clock, the overall time spent in transmission and reception more or less remains the same. Consequently, we have a situation where we are saving on performance with little impact on energy by increasing the client speed.

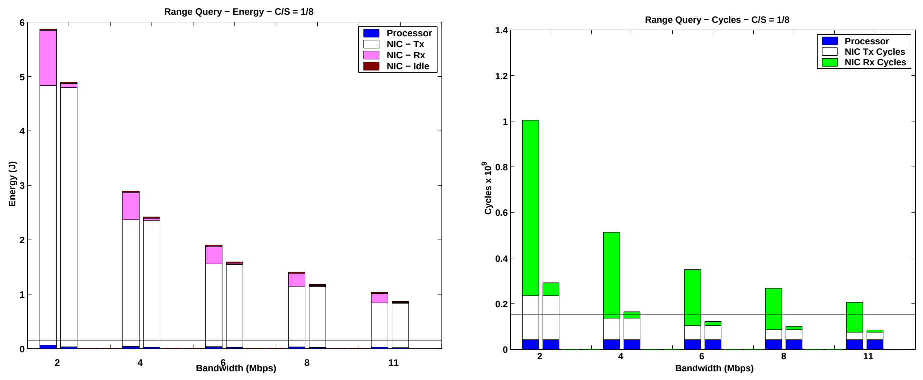
### 6.1.4 Impact of Physical Distance Between Client and Base Station

When we move from 1Km (that has been assumed until now) to a 100 m range for transmission, the power consumption drops from about 3 W to about 1 W (as given in Table 2). Here, we just show the energy graphs for the schemes with the range queries in Figure 9 (the influence on cycles is negligible). As is to be expected, with a shorter distance, the work partitioning schemes, especially those that use more transmission power (such as ”Filtering at Client, Refinement at Server”) become much more competitive. Thus, a client in a region with a higher density of base stations can transmit with lesser power, and prolong the battery energy, than a client in a region where base stations are farther apart.

(a) Fully at the Server



(b) Filtering at Client, Refinement at Server



(c) Filtering at Server, Refinement at Client

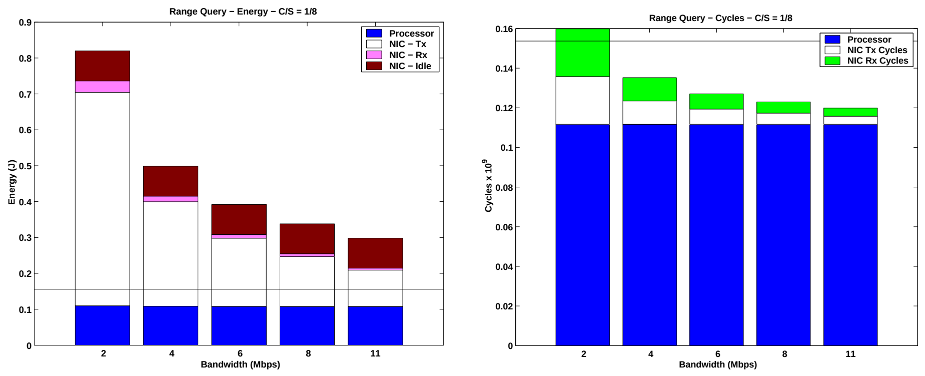
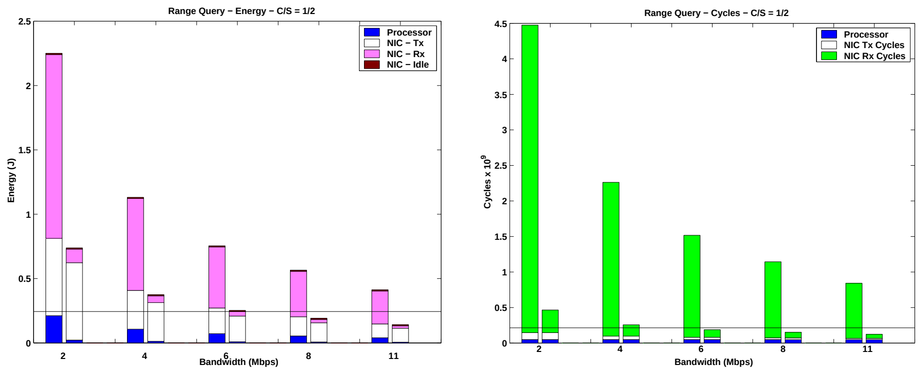
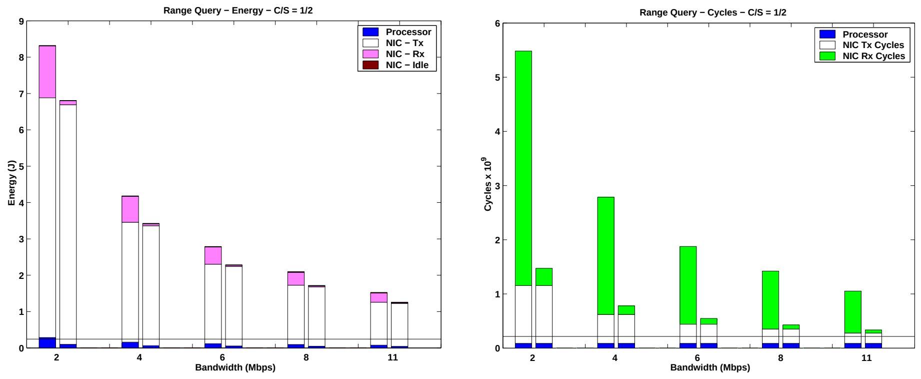


Figure 7: NYC Dataset - Range Queries. Comparing the schemes in terms of energy consumption on the mobile client, and total cycles taken for the execution. The horizontal line indicates the energy and performance of ”Fully at the Client”. The profile for energy and cycles is given in terms of what the mobile client incurs in the NIC (given separately for transmission, reception and idle) and all other hardware components that are bunched together as processor. 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.

(a) Fully at Server



(b) Filtering at Client, Refinement at Server



(c) Filtering at Server, Refinement at Client

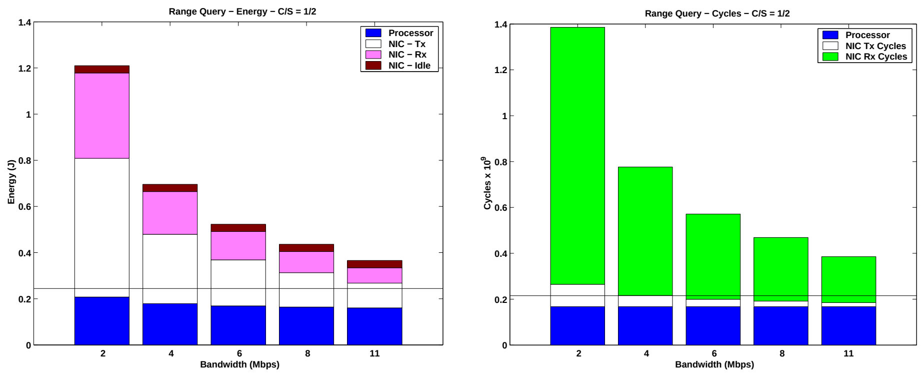
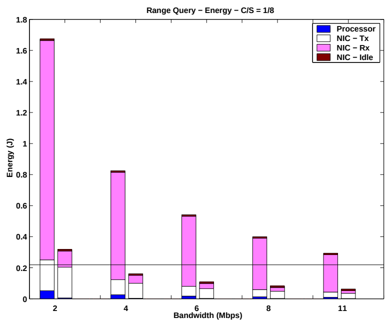
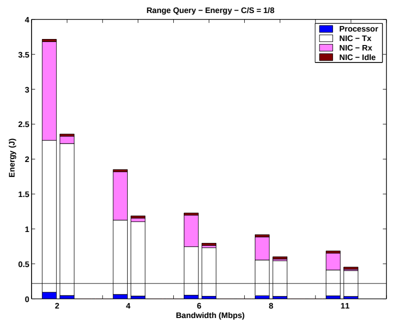


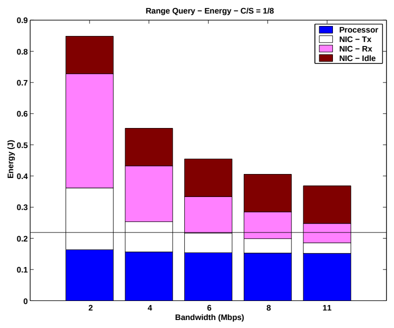
Figure 8: Studying the impact of a faster client for Range Queries. $Mhz_C = Mhz_S/2$ in this case, while it was $Mhz_S/8$ in Figure 5. The new client clock speed is used to denote the cycles in the performance graphs.



(a) Fully at Server



(b) Filtering at Client, Refinement at Server



(c) Filtering at Server, Refinement at Client

Figure 9: Energy Consumption with 100m Physical Distance between Client and Base Station. Compare this to the 1 Km distance used in Figure 5.

## 6.2 Insufficient Memory at Client

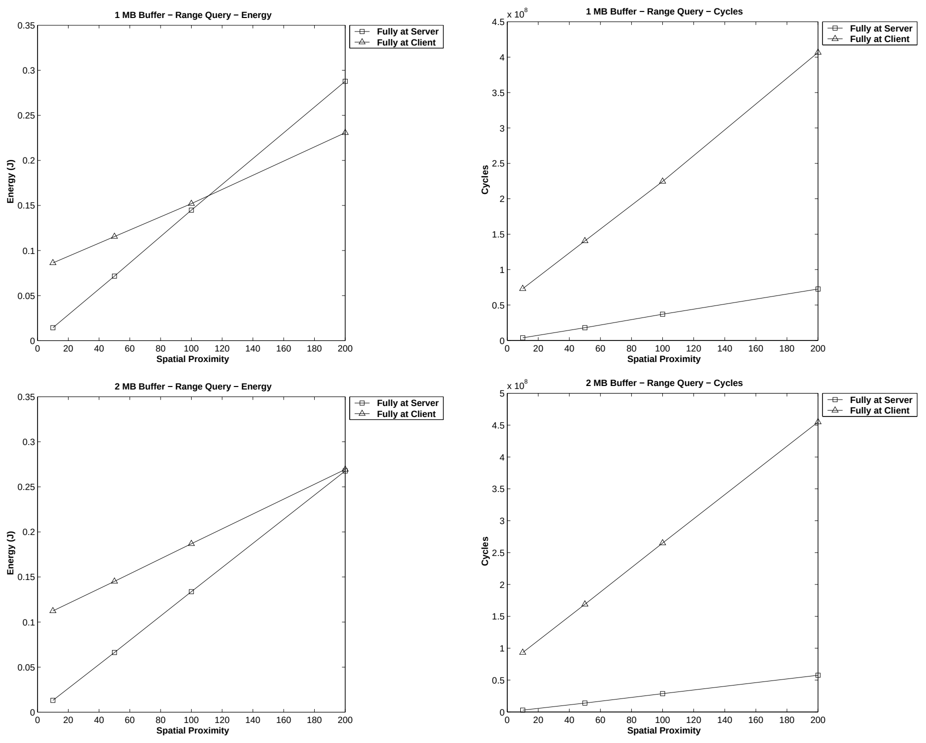


Figure 10: Insufficient Memory at Client - Range Query

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 10. 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 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. The resource-constraints, limited battery energy and memory in particular, make it challenging to write applications that make the modus operandi or access environment (whether at the desktop or on the road) insensitive to performance. A previous study [2] 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 serious 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 (we call this the ”fully at client” scheme). It is not clear if this is feasible in the first place (because of limited memory) for reasonably large datasets. Further, one may get better performance and energy savings by offloading some or all of the work to a resource-rich server across a wireless network. In this paper, we have addressed this important 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.

This work (query) partitioning problem in mobile databases and wireless communication is an issue that has been little explored, and by conducting the first in-depth examination into this problem, we have been able to identify several useful details that can be invaluable for future mobile application development:

- It is important to find points in the program for work partitioning, i.e. what to offload to the server and what to do on the client. We do not want to do this at arbitrary points in the execution since that involves shipping a lot of state information back-and-forth and can make programming difficult. With respect to spatial query processing, we showed four different alternatives for performing the work based on where the filtering and refinement steps are done. It would be useful to also exploit parallelism between client and server executions, though we have not explored this issue here.
- Apart from partitioning the work, it is also important to keep the data close to the computation that requires it. Else, a high penalty is paid in both performance and energy for data transfers. There could be situations where one could simply download all the data (and even the index) to the client, either before going on the road, or incur a one-time cost for downloading this information. It should be noted that receiving is much less expensive than transmission from the energy viewpoint. Further, the receive cost can be amortized by the savings over several queries on that dataset. Even if not all the dataset can fit in the memory, one should try to download whatever data can fit, based on the spatial region that the user may be interested in.
- Wireless communication is expensive. If the amount of work to be done is very small (as in point queries), it is better - for both performance and energy - to do all of it on the mobile client as long as the data is accessible locally. One should go to the server only if the client does not have the data.

- Work partitioning between client and server becomes very important as the amount of work increases (when moving from point to range queries). In such cases, it makes more sense to perform the refinement, which is usually more time consuming for spatial queries on memory resident data, at the server rather than on the client. Hence, one could opt to do everything at the server or to do filtering at the client and refinement at server. We find visible evidence of energy and performance savings with work partitioning for reasonable wireless bandwidths that are available today.

- Apart from the amount of work, the selectivity of the query also influences the effectiveness of work partitioning. As the selectivity decreases, the amount of data that needs to be moved back-and-forth also decreases, favoring the work partitioning schemes over doing all operations locally on client.

- The other factors favoring work partitioning are the relatively slow speed of the mobile client, and shorter transmission distances. Current server CPU speed offerings are in the 2 GHz range while the handheld/PDA CPUs are in the 200 MHz range. With time, it is possible that the gap between the two increases, favoring offloading of work. Shorter transmission distance in a region dense with base stations can help the client transmitter consume much less power.

- Energy and performance do not necessarily go hand-in-hand. There are several situations where we found performance savings by an approach, coming at a higher energy cost (e.g. Figure 5 (b)). There are also situations where there are energy savings, coming at a performance penalty (e.g. Figure 10). There are several factors governing this interplay, and the most important of these (that dominate over most others) are the energy and performance costs of communication. We find that the energy overhead that one incurs for wireless communication (particularly in transmission) plays a more dominant role than its performance overhead.

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. This includes work partitioning techniques that can exploit parallelism and pipelining, consideration of other spatial queries, incorporation of broadcast (widely shared information) into our framework, and examining issues when data is frequently modified (and the latest copy needs to be obtained from server). We intend to incorporate the lessons learned from these studies to develop an actual mobile/ubiquitous SDBMS implementation.

## References

[1] R. Alonso and H. F. Korth. Database System Issues in Nomadic Computing. In *Proceedings of the ACM-SIGMOD Conference*, pages 388–392, 1993.

[2] N. An, A. Sivasubramaniam, N. Vijaykrishnan, M. Kandemir, M.J. 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.

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

[4] N. Beckmann, H-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for points and Rectangles. In *Proceedings of the ACM-SIGMOD Conference*, pages 322–331, 1990.

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

[6] R. Cerqueira, C.K. Hess, M. Román, and R.H. 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.

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

[8] M. Cherniack, M. Franklin, and S. Zdonik. Data Management for Pervasive Computing - Tutorial at the International Conference on Very Large Data Bases (VLDB), September 2001. http://www.cs.berkeley.edu/ franklin/Talks/CFZvldb2001_6up.pdf.

[9] 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.

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

[11] J. G. Griffiths. An Algorithm for Displaying a Class of Space-filling Curves . *Software - Practice and Experience (SPE)*, 16(5):403–411, May 1986.

[12] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM-SIGMOD Conference*, pages 47–57, 1984.

[13] 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.

[14] 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.

[15] 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.

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

[17] I. Kamel and C. Faloutsos. On Packing R-trees. In *Proceedings of the ACM Intl. Conf. on Information and Knowledge Management(CIKM)*, pages 490–499, Washington, DC, 1993.

[18] 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.

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

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

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

[22] R. C. Nelson and H. Samet. A Population Analysis for Hierarchical Data Structures . In *Proceedings of the 1987 ACM SIGMOD International Conference on Managment of Data*, pages 270–277, San Francisco, California, May 1987. ACM Press.

[23] J. M. Patel. *Efficient Database Support for Spatial Applications*. PhD thesis, University of Wisconsin-Madison, 1998.

[24] N. Roussopoulos et al. Nearest Neighbor Queries. In *Proceedings of the ACM SIGMOD*, pages 71–79, 1995.

[25] N. Roussopoulos and D. Leifker. Direct Spatial Search on Pictorial Databases Using Packed R-Trees. In *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, pages 17–31, Austin, Texas, 1985. ACM Press.

[26] B. Seeger and H-P. Kriegel. The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems. In *Proceedings of the VLDB*, pages 590–601, 1990.

[27] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 507–518, Brighton, England, September 1987. Morgan Kaufmann.

[28] 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.

[29] 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.

[30] 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.