DSearch: Distributed search for a personal area network

Garrett Brown, Daniel Fabbri, Brett Higgins and Azarias Reda
{garretto, dfabbri, brettdh, azarias}@umich.edu
University of Michigan, Ann Arbor, MI

Abstract

An increasing amount of data is being stored on mobile devices with growing storage capacity and functional specializations. As a result, searching through a user's distributed data set effectively is crucial. Previous search architectures tuned for single, stationary devices are not effective at managing the challenges associated with querying data across heterogeneous machines. These designs do not consider the complex set of constraints and challenges in the distributed search domain. We propose a distributed architecture, DSearch, to manage the complexities of a mobile data set to improve query performance across all the devices in a user's personal area network. First, we provide a light-weight infrastructure that can efficiently organize and search a set of devices. Second, we develop a membership system to manage the dynamics of multiple devices in a network. Third, we analyze three search index replication schemes to improve query performance. We developed the DSearch distributed search architecture and evaluated its performance.

1. Introduction

Searching through a user's distributed data set effectively is crucial. User content is increasingly stored on multiple digital devices. In fact, it has been estimated that 55% of all digital information resides on personal computers [11]. Furthermore, individuals continue to purchase new cell phones, laptops and hand held devices with ever growing storage capacities and functional specializations. Previous search architectures tuned for single, stationary devices are not effective at managing the challenges associated with querying data across heterogeneous machines. To support the changes in storage patterns, alternative search organizations are necessary. This paper proposes a distributed search architecture, *DSearch*, for multiple devices in a user's personal area network.

Conventional desktop search and distributed file systems have relied on practically unlimited resources to organize and search user content. These designs do not consider the complex set of constraints and challenges in the distributed search domain. Specifically, these systems take for granted devices' physical location, connection capabilities and intermittent periods of connectivity. Also, complex file systems assume the presence of a particular operating system running on a high-powered processor, in contrast to extremely heterogeneous mobile devices that have limited processing power. Given these constraints, we examine the best way to replicate search indexes within a user's personal area network for improved query performance.

We propose a distributed architecture, DSearch, to manage the complexities of a mobile data set to improve query performance across all the devices in a user's personal area network (PAN). First, we provide a light-weight infrastructure that can effectively organize and search a set of devices. Because of mobile devices' limited computation abilities, full-fledged data indexing mechanisms are not practical. Second, we develop a membership system to manage the dynamics of multiple devices in a PAN that records the current set of active devices and distributes information to the group.

Third, we examine three search index replication schemes to improve query performance. In the basic, no replication architecture, queries are sent to each active device in the PAN. Every device in this configuration searches its own content and responds with a list of matching files. The basic design limits search performance since the slower devices are queried and must be waited on for complete query aggregation. A centralized architecture improves on the basic design by replicating all search indexes to a coordinator, which is assumed to be always-on, although this does introduce a single bottleneck and point-of-failure. For each query, only the coordinator is sent a request and computational slower devices are bypassed. Lastly, we provide a device-based replication scheme that allows each device to select which other devices' search indexes to store locally based on query latency. We vary the number of replicas that can be stored at each device and examine the query time for all the designs.

This paper makes the following contributions:

- **Light-weight search infrastructure.** Python based application architecture that runs on desktops, laptops and mobile devices.
- **Dynamic membership management.** System to manage the arrival, departure, and intermittent connectivity of devices in the personal area network.
- **Search index replication for improved query performance.** Three replications architectures (no replication, centralized replication, device-based replication) tested and evaluated for query performance.

In Section 2 we provide a brief summary of the background in search systems and discuss related work. We go into more detail on DSearch itself in Section 3, and describe the implementation in very granular detail in Section 4. We layout a graphical model of our replication optimization scheme in Section 5 and evaluate our system in Section 6. We give our conclusion in Section 7.

2 Background and Related Work

Personal file search systems are now common; Windows Desktop Search [1], Google Desktop [9], Apple's Spotlight [15], and the open source project Beagle [5] are several examples. Each maintains an up-to-date index of the user's content, allowing them to quickly find matching content based on a search query. However, as users tend to own multiple devices, content is often spread across those devices, and there is no unified interface from which to search the user's "cloud" of devices.

A distributed search system is one that allows a user to query a set of devices from any one of those devices and retrieve search results, identifying files and the devices on which they are located. Each device is responsible for the content it "owns" and therefore should maintain its own index. Such a system would need to have mechanisms for devices to join and leave the system, to locate other devices in the system and exchange messages with them, and to distribute query requests and aggregate query results. The distributed search system is similar to the aforementioned desktop search systems in that each device can be seen as an independent search entity, but the distributed system must provide means to share the individual devices' search results among all members of the system.

Various approaches have been applied to distributed search thus far. For instance, some work has been done on expanding the traditional distributed file system to effectively address needs of more consumer electronics devices [12]. Others have approached the problem from the other end, starting with the intermittently connected mobile devices and trying to maintain or impose some overlaying structure. Flinn and Anand proposed PAN-on-Demand, a self-organizing overlay network scheme which utilizes differing radio capabilities and device heterogeneity for the ultimate goal of achieving excellent power usage [2]. Differing modes of membership have also been proposed and some studies have focused on the cost differences between real-time discovery versus connection maintenance [2]. Others have focused on the potential performance gains of using multiple interfaces (e.g., Bluetooth and WiFi) simultaneously while operating at the application layer [3] or by modifications at the transport layer [10].

Similarly, researchers have also proposed various ideas for distributing data to multiple mobile devices [16] as well as the potential performance benefits from caching data on a device-by-device basis [11]. Consideration has been given to the importance of how data is stored or distributed across a p2p network in order to preserve locality of data while still maintaining effective and scalable query throughput [13]. Other approaches have aggregated a user's devices into a single addressable virtual device focusing on higher level "person-level routing" [6, 4], while others have focused on lightweight development frameworks [14].

3 DSearch

An increasing amount of data is being stored on mobile devices with growing storage capacity and functional specializations. Similarly, users now own an increasing number of devices, dividing their content among them. As a result, the ability to search through a user's mobile data set becomes important. Unlike desktop search, however, searching a network of mobile devices presents a different set of constraints and challenges due to devices' locational diversity, varied connection capabilities, intermittent periods of connectivity and device heterogeneity.

First, because of the limited computational capacities of these devices, full fledged data indexing mechanisms are not practical. Instead, there is a need for a light-weight indexing infrastructure that can effectively target these devices. Second, personal devices are by nature mobile, and any system that searches among these devices needs to consider intermittent availabilities and changes in the location of devices. This calls for a technique for managing the personal area network (PAN) and identifying and maintaining members of the group. These and other challenges make distributed mobile search an interesting domain to work in.

We developed DSearch, a light-weight distributed search application. DSearch manages devices leaving and joining the network. We provide a basic infrastructure to search content on all the connected devices and provide three search index replication schemes to evaluate their impact on query performance.

4 DSearch Implementation

DSearch has a Python client that runs on each device in the PAN. Our system is platform independent and only requires that Python and SQLite be installed on each device. The implementation operates on Mac OS X and Linux based operating systems. One device acts as the coordinator of the PAN. This device is assumed to be on most of the time. Various settings control the type of replication scheme, the port to listen on and the address of the coordinator. When DSearch is started at a device, the system registers with the coordinator and is assigned an identification number, which is distributed to all other active members along with other member information. The coordinator keeps track of devices as they leave and rejoin the network.

The owner of the device interacts with DSearch through a command-line interface. This interface allows the user to specify directories to be indexed to satisfy future queries. Queries are initiated through the command-line interface. The DSearch architecture multiplexes the query to the other members of the PAN depending on the replication scheme.

The DSearch implementation is divided into four categories: communication systems, data management, membership management, and searching infrastructure. The communication system provides the networking interfaces for DSearch. The data management system indexes local data and stores it for future queries. The membership module manages the dynamic joining and leaving of members in the group. The search infrastructure implements the basic query system as well as three search replication schemes.

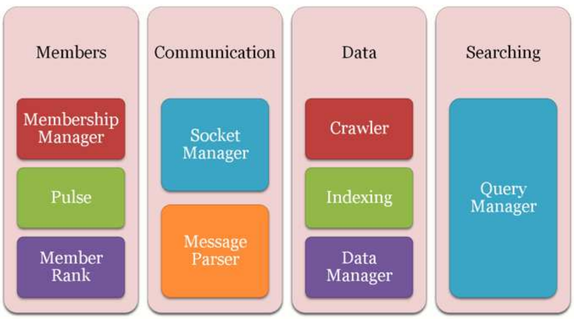

Figure 1. DSearch High Level Design

4.1 Data Management

The Data Manager provides a mechanism to examine file content and provides a search facility. Search systems usually use some variation of an inverted index, in which keywords map to files containing those words. We take a similar approach in DSearch. Since our system involves several devices storing different content, our index maps keywords to $(memberId, filepath)$ pairs, where a $memberId$ (assigned by the coordinator) identifies a member uniquely within the system.

4.1.1 Index Manager

The IndexManager maintains a list of root directories that are currently being indexed. On program startup, the IndexManager spawns a thread and scans through each of the directories in the list recursively, looking for keywords and adding mappings to the index. The user can add and remove directories through the command-line interface, and the index is immediately updated in response to those commands. In the absence of add and remove requests, the IndexManager thread will periodically wake up and refresh the index by scanning all the root directories and repopulating the index structure.

4.1.2 Data Manager

The DataManager module implements the backend data structure for our indexing mechanism using a SQLite3 database. The DataManager abstraction includes methods to insert (word, file) matches as the indexer processes files. In addition, the DataManager provides a method to construct an XML message, encapsulating the index structure it stores, to be sent to other members upon request. This functionality is used by our index replication stragegies.

4.1.3 Index Shipping

In order to allow members to request and receive search index replicas, the IndexManager keeps a list of *subscribers*, or members who have requested this member's index. Upon receiving a $requestIndex$ message, the IndexManager adds the requester to the subscribers list if it is not already present and responds by sending an XML message containing its index. From then on, whenever the member updates its index (whether periodically or due to a user command), it will re-send its index to all of its subscribers. Though this is more costly than sending incremental index updates, it also greatly simplifies the message-passing semantics, since each index update sent to a subscriber is idempotent.

4.1.4 Consideration of Tradeoffs

As we designed our search and indexing mechanisms, some clear tradeoffs presented themselves. First, there is a significant computational cost involved in crawling through directories and extracting keywords from files. Initially, this cost is compulsory; the first time a directory is added to the index, the content has never been seen before and must be scanned and read completely. After the initial scan, however, it is sufficient to only re-scan files that have changed since the last scan. Our observation is by updating the index (with a full re-scan) frequently, we increase the likelihood that searches have the most up-to-date view of the indexed content, but with the computational cost incurred by frequent scans. Also, as the size of the total content being indexed on a mobile device grows, the database queries involved in finding keyword matches become more costly. It is at this point that shipping indexes to more well-provisioned (e.g. 2GHz, 2GB RAM, AC power vs. 400MHz, 128MB RAM, battery power) devices becomes attractive, since avoiding executing the query at the mobile device will save time and power, especially when the rate of queries is much higher than the rate of content update.

Second, there is a tradeoff between the size of the index and the robustness of the search. Currently, we keep a count of the number of times each keyword appears in each file, but we do not store any positional data (for example, to allow searching for phrases). Whereas most personal file search systems are first optimized for speed, a search system intended for deployment on mobile devices must carefully consider how much storage to spend in return for better query results. Given our simplistic indexing implementation, we defer a rigorous exploration of this tradeoff to future work, though it is worth noting that our current indexing mechanism requires a 924KB database to index the content of 22 files totaling 26MB in size.

4.2 Membership Protocol

As a distributed search system, DSearch needs to manage its members in an efficient manner. One of the basic assumptions that we made is that one of the members will act as the coordinator and the other members will have to register with the coordinator to be part of the system. Any one of the members could be the coordinator, although it is benefitial to have a usually on device assume this role. The membership protocol is composed of two modules, Memberhsip Manager and Pulse.

After one device is started as the coordinator for the system, other devices can register with it. When the coordinator receives a register request from a device, it looks up its members list and assigns the new device a unique ID, which is broadcast to the rest of the system along with the new device's credentials. With this information, devices can set up TCP connections to other members in the group.

Pulse is the activity manger of the system. It maintains a current view of the system from a particular device rather than the global state kept in Membership Manager. The basic mechanism used to achieve this is heartbeat messages sent periodically from the members to the coordinator. A member is allowed a few cycles of "grace" to send its heartbeat to the coordinator before deemed inactive. This value, set by the administrator of the system, determines how long a coordinator waits before removing a member from the active list, and hence it also determines the upper bound on how long the active member list could be stale on any particular device. Setting the grace to be higher will give member a longer duration to reconnect to the system with out having to go through the registration process again. This might be especially important in mobile devices where the connections are likely to be intermittent. On the other hand, setting the grace to be smaller gives a more current view of the system at any given time.

Pulse propagates the active list to the members in one of two modes. The default mode is event triggered, in which the active list is distributed to members whenever an event occurs on the coordinator that changes the current active list; these are mainly leaves and joins. The other mode is an on-demand mode in which a member receives the active list only when needed, either to make caching decisions or to send out messages. When a member receives the active list, it also gets a lease time on the current active list, which is as long as the grace. If the list is any older, a member will have to request it again. Setting the lease to be equal to the grace makes sense because that is also the earliest a coordinator publishes the deletion of a device from the members list.

The on-demand mode is most efficient when the join/leave rate is much higher than the rate of requests for the active list. By requesting the active list only when needed, members avoid getting unnecessary updates that are potentially going to be out of date soon. On the other hand, when there is low join/leave activity on the system, event-triggered updates is more appropriate because the list distributed is potentially going to be useful for a while.

4.3 Query Management

The Query Managemen module is responsible for distributing queries to all active members of the DSearch network as well as merging and aggregating the individual responses. With the default no-replication scheme, the query is sent to each active member of the network. On the other hand, when utilizing the centralized replication scheme, a single query is sent to the coordinator. Lastly, with the device-based replication scheme, queries are distributed strategically to exploit the more efficient processing capability or lower network latency exhibited by certain members of the network. Regardless of the replication scheme, every member of the current system is queried either directly or indirectly and its response is aggregated.

As a side note, there is a querying capability to allow the user to query only a specific device. This is useful in situations where the user is only interested in files residing on a laptop or MP3 player rather than all devices. This feature

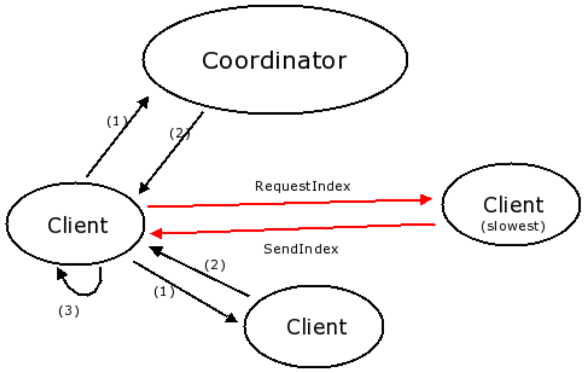

Figure 2. Querying in device-based replication mode

meshes as expected with the various replication schemes as only the member responsible for the device specified is queried.

Once all queried devices have responded, the results are ranked for display. Our current ranking scheme orders the results by (*Number of Search Term(s) Found in File*) / (*Total Number of Terms in File*). More complex ranking schemes are certainly feasible, but this metric proved simple and useful enough to satisfy our current requirements. Once the results have been ranked they are displayed to the user through the console as *MemberId — Filepath — Frequency / Total Terms*.

4.4 Replication Schemes

Index replication is used in DSearch as a method for improving availability and performance. DSearch replicates search indexes of devices at various places in the system. There are three modes of operation implemented in DSearch. These modes are device-independent, which means the selection affects only a particular device, and a PAN might be composed of devices running different modes of replication.

The first mode implements no replication. Each device is responsible for its own index, and whenever a member needs to query the contents of another member, it simply sends a direct network message to it. This is the most obvious way of doing things, and its biggest advantage (besides being simple) is that it always produces up-to-date search results. On the other hand, it generates large amounts of network traffic, and introduces search latency from the slower devices.

The second mode is a coordinator replication mode. This mode assumes a coordinator that has a higher network bandwidth and storage capacity. If the coordinator is running this mode, whenever a device registers, it is informed of the mode and requested to ship its index to the coordinator. This is potentially an expensive operation upfront. However this approach might make sense when the coordinator is relatively faster and queries are done at a consistently high rate. In these cases, the up-front cost will be amortized quickly. Once a coordinator has all active members' indexes, all searches are performed at the coordinator.

The local replication mode (Figure 2) is a more involved replication scheme than the previous two and is implemented using the Member Rank module. Whenever a device starts up, the user provides a hint about the memory space available for locally caching other device indexes. The Member Rank module sends messages to the active devices requesting what indexes they already have cached. It then uses the round trip time to rank the various members according to the latency. Because DSearch waits for all devices to respond to queries (either with results or some form of notification of a process crash, such as a socket error or timeout), the entire query execution time is as slow as the slowest of the members. So, ideally a member benefits most by replicating the slowest of the devices locally.

However, this is complicated by two issues. First, there is a limit on how many devices each member can replicate, and second there might be faster devices that have already replicated slower devices' indexes, so that it might be faster to use those existing replicas than creating a new one, and instead replicate some other device's index. This replication analysis problem is an NP-complete problem that is most similar to the Weighted Directed Dominating Set problem (Section 5). DSearch implements an approximation algorithm which puts into consideration these various factors. The algorithm first significantly prunes the search domain and then greedily suggests the best devices to replicate at each step.

A replication mechanism tailored to the capabilities of individual devices is ideal in a heterogeneous system of devices in which various members have different storage, latency and bandwidth limitations, and has been shown to be very effective [2]. In those cases, local replication tries to find the optimal replication scheme given the current state of the system. It is also interesting to note that under the assumptions for the coordinator replication mode, in which the central device is faster and has plenty of storage space, local replication mode converges to the coordinator replication mode when we are able to replicate all members' indexes locally. By paying a limited upfront cost, a device can minimize its query response time, and upfront cost will be amortized quickly.

In any replication system, it is important to consider consistency among replicas. DSearch uses a primary-based consistency in which one device owns a definitive copy of its own index. Also, we notice that DSearch has a unique feature in that only the owner of an index ever writes to

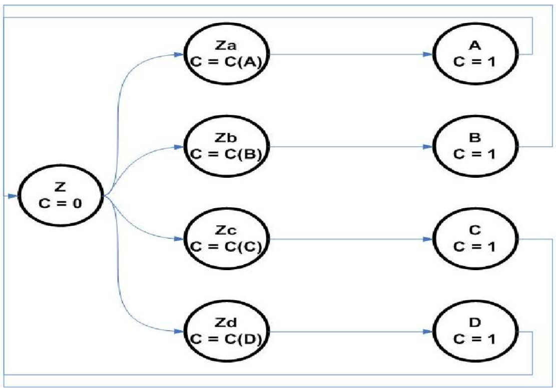

Figure 3. Basic DSearch Graphical Representation

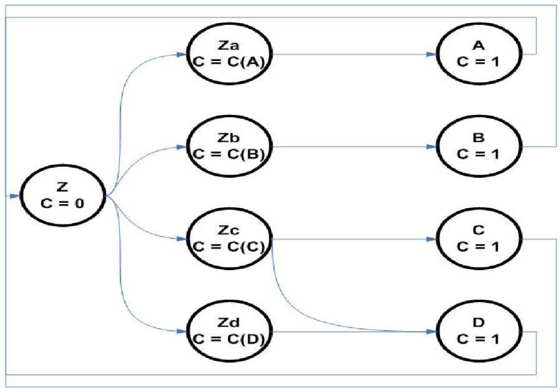

Figure 4. Same as Figure 3, but in this example, C stores a replica of D's index.

it, and all other members simply read from it. Because we have an authoritative replica, implementing consistency is straightforward. We achieve consistency using subscriptions to the owner of a copy. Whenever a device requests some other member's index, it also subscribes to be notified about any changes in the index. So, after a device re-crawls its folders, it sends its index to all its subscribers. The window of stale indices is then reduced to the time it takes for a network update message from the source to reach subscribers. We believe this is sufficient to get current results in a personal area network.

5 Graphical Model

Selecting the optimal replica to store locally is essential for minimizing query run time. The objective is to select the replica(s) that will maximize the performance gains. Our current implementation uses various techniques such as graph pruning to select which search indexes to store. We present a graphical model that reduces the selection problem to a directed, weighted dominating set problem.

Given a graph, each device must select the optimal replica to store locally. We specify a constant k that determines the maximum number of replicas that can be cached, where k is representative of the device's memory capacity. When a new member joins the network, it subscribes to at most k other devices and receives their search indexes. Each device makes this selection through a calibration technique that ranks the time to get to every device in the PAN and also accounts for other search indexes already replicated to other nodes.

We present a graphical model to assist in solving of this optimization problem. When a device joins the PAN, it creates a graphical representation of the network. Before the graph can be created, all the members of the PAN are ranked based on query latency. Given four members $\{A, B, C, D\}$ with latency costs $1 < C(A) < C(B) < C(C) < C(D)$ and a new node Z with a cost of zero, we create the graph seen in Figure 3. This graph assumes that no replicas have been sent up to this point in time.

The graph is divided into three columns of nodes. The left column contains the new member Z, which has a directed edge to $\{Za, Zb, Zc, Zd\}$. These edges represent device $Z's$ ability to send a query to node X through node Zx. The right column represents the possible replicas that can be stored locally. Our construction only allows k of these nodes can be selected at a time. An edge is added from node Zx to node Y if X has $Y's$ search index stored locally.

To select the optimal replica, we solve the directed, weighted dominating set optimization problem. A dominating set is a set of nodes in the graph such that every node in the graph is either in the dominating set or is connected to a node in the dominating set. For a directed graph, this implies that if A is in the set and there is an edge from A to B, both A and B are covered; however, if the edges goes from B to A, B is not covered by the set. We also must reduce the total cost of the dominating set, which corresponds to the query time. Notice that Z is always added to the set since it has no cost and the cost to store a replica is less than the cost to send a query.

For the case of $k = 0$ in Figure 3, no replicas can be stored and a query must be sent to each device. The dominating set for this network contains $\{Z, Za, Zb, Zc, Zd\}$.

For the case of $k = 1$ in Figure 3, since Zd has the highest cost, it is best to choose the dominating set of {*Z, Za, Zb, Zc, D*}.

For the case of $k = 1$ in Figure 4 where C already has $D's$ search index stored locally, it is not optimal to store $D's$ search index since it can be accessed more efficiently through C. Therefore, the optimal set is $\{Z, Za, Zc, B\}$.

Note that storing C is not optimal since node D would then need to be queried, adding the highest possible cost.

The dominating set decision problem is NP-complete [7, 8]. Instead of solving for the optimal solution, we apply approximation techniques that greedily select the highest cost, un-covered replicas to store. This algorithm runs in $O(N)$ time, where N is the number of members in the PAN.

6 Evaluation

6.1 Evaluation Methodology

To analyze DSearch, we preformed a set of experiments to compare the performance of the three replication schemes. We deployed DSearch on multiple Linux and Mac OS X desktops and laptops as well as on one Nokia N800 handheld device. The desktops and laptops had processors clocked from 1.5-2 GHz and had 1-2 GB of main memory compared to the N800's 400 MHz processor and 128 MB of main memory. We measured the total query execution time, defined as the time for all query responses to be received and aggregated at the querying device.

We performed the following tests. First, we measured the query performance as the number of members in the PAN increased while no data was indexed at each device. This test measures how the network performance of the system scales as new members join. We also collected network traffic to measure the growth of the system's bandwidth consumption as the PAN size increases. Second, we gathered performance data for the three replication schemes. We tested the no-replication method, the centralized approach and the device-based scheme and compared their performance. For each of these tests, we indexed a test set of 22 files that included MP3, text and PDF files totaling 26MB in size and requiring an 924KB database. The handheld had a smaller set of files (approximately 688KB) since it takes orders of magnitude longer to index data with the slower processor (e.g. 20 minutes vs. 30 seconds for the same data).

6.2 DSearch Query Performance

First, we analyzed how the DSearch query architecture scaled as devices joined the PAN without any data indexed on the device. Querying empty databases provides us with a bound on the query time and provides insight into the networking costs of a distributed system. When the PAN did not include the handheld device, the query times were extremely small (e.g., 0.008s) and negligible due to the fast interconnects of the LAN. When the handheld was included, query times increased to approximately 0.1 seconds,

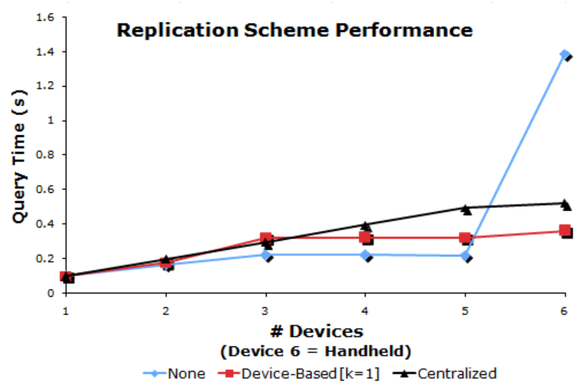

Figure 5. Replication Scheme Performance

demonstrating the impact of wireless networks on query performance.

Next, we indexed the data stored on each device and executed queries for the three replication schemes. Figure 5 shows the performance plots for the three designs. The basic, no-replication method issues queries to each device in parallel. As more devices are added, the query time increases because of the network latency and extra time required to aggregate queries from multiple sources. Note that the query performance remains almost constant when the handheld is not included and when more than one device is in the PAN. Query performance does not vary since each query is issued in parallel and the devices have similar processors. When the handheld joins the PAN, the query time increases by a factor of seven, due to the N800's slower processor and large wireless network latency.

The centralized and device-based replication schemes improve the overall query performance when the handheld is included in the PAN. For the centralized approach, a single query is sent to the coordinator, which then searches in its database for all matching files from throughout the PAN. Since the coordinator contains the aggregate databases from all the devices in the PAN, searching at the coordinator is comparatively slower than the no-replication method. Therefore, for queries not including the handheld, the basic approach outperforms the centralized method, where query time increases almost linearly with the number of devices. However, the centralized method more efficiently processes queries that include the handheld since there is no need to directly query the slower device. Furthermore, the delay from sending indexes is amortized over time due to the improved query performance.

The device-based replication scheme improves on the weaknesses of the centralized model. A drawback of the centralized method is that query time grows linearly with the number of devices, since more data must be searched at the coordinator. We observed from the no-replication

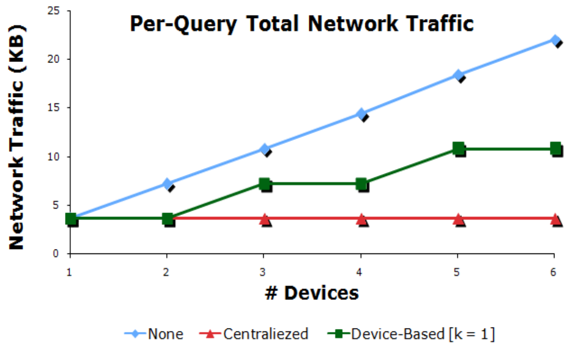

Figure 6. Network Traffic Generated

model that query time remains constant when multiple, similar performing devices are queried in parallel. Considering this observation, the device-based replication scheme stores a subset of the replicas locally. From Figure 5, we find that the coordinator and device-based methods perform similarly when up to three members are in the PAN, but as more members are added, the device-based replication model maintains a constant performance in contrast to the worsening centralized method. Our observations also hold when the handheld is included in the PAN.

6.3 Alternative Metrics

Shipping search indexes from device to device to improve query performance utilizes more network bandwidth than the no-replication model. When we transfer a search index, we send an XML message containing all the relevant file information stored in the database. For our tests, this message was approximately 140 KB in contrast to approximately 4 KB of data to send and receive query responses from each node with no replication for the tested query. Since devices are often left on for long durations of time, we believe sending the larger data stream over period of time is an acceptable tradeoff for improved query performance. Figure 6 shows the network traffic generated for each replication scheme as the number of devices in the PAN increases from 1 to 6.

7 Conclusion

We have presented DSearch, a system which enables users to search files distributed across their set of personal devices. We developed three different index replication mechanisms and evaluated their performance, showing that careful index replication can improve query performance and scalability. We feel that DSearch will be a useful framework for future endeavors in managing content in many personal distributed networks with mobile devices.

References

[1] Windows search. http://www.microsoft.com/windows/desktopsearch/.

[2] M. Anand and J. Flinn. Pan-on-demand: Building self-organizing wpans for better power management. Technical report, 2006.

[3] G. Ananthanarayanan, V. N. Padmanabhan, L. Ravindranath, and C. A. Thekkath. Combine: leveraging the power of wireless peers through collaborative downloading. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 286–298, New York, NY, USA, 2007. ACM.

[4] G. Appenzeller, K. Lai, P. Maniatis, M. Roussopoulos, E. Swierk, X. Zhao, and M. Baker. The mobile people architecture. Technical Report CSL-TR-99-777, 1999.

[5] Main page - beagle. http://beagle-project.org.

[6] C. Carter and R. Kravets. User devices cooperating to support resource aggregation. In *Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on*, pages 59–69, 2002.

[7] S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[8] M. Garey and D. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman.

[9] Google desktop. http://desktop.google.com.

[10] K.-H. Kim and K. G. Shin. Improving tcp performance over wireless networks with collaborative multi-homed mobile hosts. In *MobiSys '05*, pages 107–120, New York, NY, USA, 2005. ACM.

[11] C. Lindemann and O. P. Waldhorst. A distributed search service for peer-to-peer file sharing in mobile applications. In *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*, Washington, DC, USA, 2002. IEEE Computer Society.

[12] D. Peek and J. Flinn. Ensemblue: integrating distributed storage and consumer electronics. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 219–232, Berkeley, CA, USA, 2006. USENIX Association.

[13] T. Scholl, B. Bauer, B. Gufler, R. Kuntschke, D. Weber, A. Reiser, and A. Kemper. Hisbase: histogram-based p2p main memory data management. In *VLDB '07*, pages 1394–1397. VLDB Endowment, 2007.

[14] K. Senthivel, S. Kalasapur, and M. Kumar. Person: A framework for service overlay network in pervasive environments. pages 671–682. 2007.

[15] Spotlight overview. Technical Report 2006-04-04, Apple Corp., Cupertino, CA, 2006.

[16] P. Xuan, S. Sen, O. Gonzalez, J. Fernandez, and K. Ramamritham. Broadcast on demand: efficient and timely dissemination of data in mobile environments. In *Real-Time Technology and Applications Symposium, 1997. Proceedings., Third IEEE*, pages 38–48, 1997.