Improving Mobile Database Access Over Wide-Area Networks Without Degrading Consistency

Niraj Tolia*, M. Satyanarayanan, and Adam Wolbach

January 2007
CMU-CS-07-100

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA

This research was supported by the National Science Foundation (NSF) under grant number CCR-0205266. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or Carnegie Mellon University. All unidentified trademarks mentioned in the paper are properties of their respective owners.

Keywords: Content Addressable Storage, Mobile Databases, Wide Area Networks, Replicas, Bandwidth Optimization

Abstract

We report on the design, implementation, and validation of a system called *Cedar* that enables mobile database access with good performance over low-bandwidth networks. This is accomplished without degrading consistency. Cedar exploits the disk storage and processing power of a mobile client to compensate for weak connectivity. Its central organizing principle is that even a stale client replica can be used to reduce data transmission volume from a database server. The reduction is achieved by using content addressable storage to discover and elide commonality between client and server results. This organizing principle allows Cedar to use an optimistic approach to solving the difficult problem of database replica control. For read-write workloads, our experiments show that Cedar improves throughput by 39% to as much as 224% while reducing response time by 28% to as much as 79%.

1 Introduction

Relational databases lie at the core of many business processes such as inventory control, order entry, customer relationship management, asset tracking, and resource scheduling. A key contributor to this success is a database's ability to provide a consistent view of shared data across many geographically dispersed users, even in the face of highly concurrent updates at fine granularity. Transactional semantics form an essential part of this consistency model.

Preserving consistency with acceptable performance under conditions of weak connectivity is a difficult challenge. Indeed, it has been widely accepted since the early days of mobile computing that shared data access involves a fundamental tradeoff between consistency, good performance, and tolerance of poor network quality [12]. This has led to a variety of approaches (discussed in Section 7) that relax consistency. However, failing to preserve consistency undermines the very attribute that makes databases so attractive for many applications.

In this paper, we describe a new approach to mobile database access that avoids the need to compromise consistency. In other words, we show how to achieve good client performance under conditions of weak connectivity without degrading consistency. Our critical insight is that the disk storage and processing power of a mobile client can be used to compensate for weak connectivity. We report on the detailed design, implementation, and experimental validation of this approach in a system called *Cedar*.

Cedar uses a simple client-server design in which a central server holds the master copy of the database. Cedar's organizing principle is that *even a stale client replica can be used to reduce data transmission volume.* It accomplishes this through the use of content addressable storage. The volume reduction is greatest when the client replica and the master copy are identical. However, even a grossly divergent replica will not hurt consistency; at worst, it will hurt performance. This organizing principle allows Cedar to use an optimistic approach to solving the difficult problem of database replica control. At infrequent intervals when a client has excellent connectivity to the server (which may occur hours or days apart), its replica is refreshed from the master copy. Cedar primarily targets laptop-class machines as mobile clients, but we also explore Cedar's applicability to PDA-class mobile clients. Since laptop disk sizes can be 100GB or more today, replicating the entire database is often feasible. For databases that are too large, Cedar provides support for creating a useful partial replica.

A `select` query on a Cedar client is first executed on the local replica to obtain a tentative result. Using cryptographic hashing, the client constructs a compact summary of the tentative result and sends it to the server along with the query. The query is re-executed at the server to obtain the authoritative result. The server then constructs a compact summary of the difference between the tentative and authoritative results and sends it back to the client. If the client and server are in sync, there will be no difference between the results. The further out of sync they are, the larger the difference in results. However, the difference is never larger than the size of the result that would be generated without Cedar. By applying the difference to the tentative result, the client reconstructs the authoritative result and returns it as the response to the `select` query. Since applications never see tentative results, they perceive exactly the same database behavior that they would without Cedar. `update` queries are routed directly to the server without special client handling.

Cedar's design and implementation pay careful attention to practical considerations. First, Cedar is completely transparent to client applications and hence requires no changes to them. Second, databases from different vendors can be used at the server and client. This simplifies interoperability, and allows client and server software to be independently optimized for their respective hardware resources and workloads. Third, we do not require source code access to any database.

2 Background

In this section we examine the characteristics of networks that Cedar targets and the reasons for their increasing popularity. We also briefly discuss two key technologies on which Cedar is built, content addressable storage and standardized database access APIs.

2.1 Wireless Wide-Area Networks

Millions of users today own personal computing devices that can access, query, and update data stores and databases over wireless networks. The increasing availability of wide-area connectivity options such as cellular GPRS/EDGE, EVDO, and WiMax has encouraged the notion of access to data anywhere and anytime [34]. However, many of these wireless technologies exhibit high variability in peak theoretical throughput, as shown in Table 1. End-to-end latency is also highly variable. To make matters worse, recent studies have shown that achievable throughput is often only 25–65% of the theoretical maximum [16, 18], and that large drops in throughput are seen during peak usage periods [40].

Wireless Technology	Peak Theoretical Throughput
GPRS	30 – 89 Kbit/s
GPRS/EDGE	56 – 384 Kbit/s
CDMA (1xRTT)	144 Kbit/s (downlink)
	64 Kbit/s (uplink)
EVDO (Rev. 0)	2,500 Kbit/s (downlink)
	154 Kbit/s (uplink)
EVDO (Rev. A)	3,100 Kbit/s (downlink)
	1,800 Kbit/s (uplink)
WiMax (802.16)	500 Kbit/s - 2,000 Kbit/s
	(per connection)

Table 1: Wireless WAN Technologies Prevalent in 2006–07

Wireless Wide-Area Network (WWAN) technologies remain important for the foreseeable future in spite of the growing popularity of WiFi (802.11) technology. First, WiFi coverage is limited to compact areas where dense base station infrastructure can be economically sustained. In contrast, WWAN technologies require much less dense infrastructure that can often be piggybacked on existing cell phone infrastructure. Hence, they are economically sustainable over much larger geographic areas. Second, organizational and business considerations may preclude use of WiFi.

For example, a user may not subscribe to the wireless service provider of a hotspot. As another example, corporate security guidelines may prohibit a salesman from using the WiFi coverage available at a customer site; hence unplanned use of WiFi might not be possible. Third, existing WiFi infrastructure may be damaged or unusable in situations such as disaster recovery and military operations. Rapid setup of new wireless infrastructure is feasible only if it is sparse. This typically implies reliance on WWAN technologies.

2.2 Content Addressable Storage

Cedar improves performance by discovering commonality across tentative and authoritative database results. Since these results can be large, eliding commonality can lead to a win on slow networks. Based on its success in networking [49], distributed file systems [37, 52, 53] and enterprise-scale storage systems [23, 41] we use *Content Addressable Storage (CAS)* induced by cryptographic hashing to discover commonality. As explained in Section 3.2.1, we have customized this technique to deliver satisfactory results in our context.

Like previous CAS-based efforts, we assume that real-world data is collision-resistant with respect the cryptographic hash function being used. In other words, it is computationally intractable to find two inputs that hash to the same output [36]. Trusting in collision-resistance, CAS-based systems treat the hash of a data item as its unique identifier or tag. Data then becomes content-addressable, with tags serving as codewords for the much larger data items in network transmission.

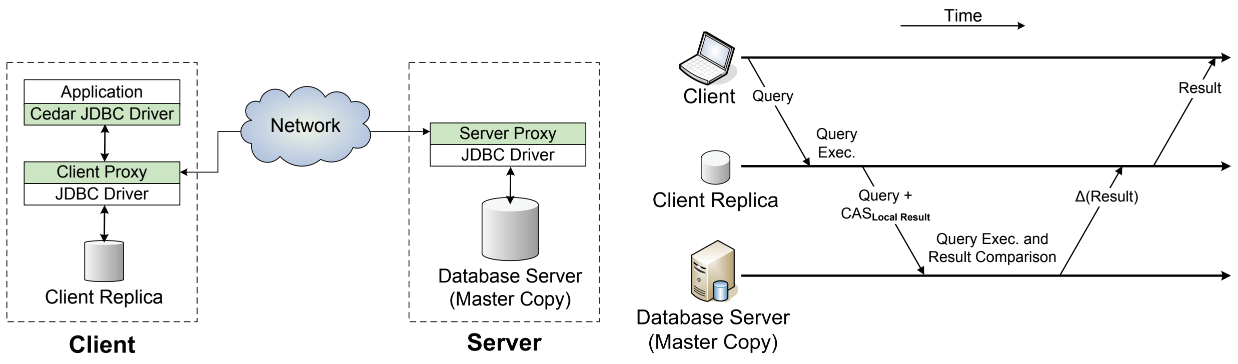
Although concerns have been expressed about the collision-resistance assumption of CAS [28], the rebuttal by Black [10] is compelling. If Cedar's hash function (SHA-1 [45] today) is broken, replacing it would be simple since Cedar only uses hashing on volatile data and never on permanent storage. While a much stronger function such as SHA-256 [46] would increase computational effort at the client and server, the new hashes would still be much smaller than the data items they represent.

2.3 Database Access APIs

Remote database access is widely supported today through *Java Database Connectivity (JDBC)* and its wire protocol antecedent, *Open Database Connectivity (ODBC)*. JDBC defines a Java API that enables vendor-independent access to databases. In other words, an application written to that API can be confident of working with any database that supports JDBC. Each vendor provides a client component called a *JDBC driver* that is typically implemented as a dynamically linked library layered on top of the TCP `socket` interface. The wire protocol between the JDBC driver and its database is vendor-specific, and typically embodies proprietary optimizations for efficiency.

3 Design and Implementation

As Cedar's focus is on improving performance without compromising consistency, it assumes that at least weak or limited connectivity is available. It is not targeted towards environments where



(a) Architecture (b) Protocol

Part (a) of this figure shows how we transparently interpose Cedar into an existing client-server system that uses JDBC. The colored boxes represent Cedar components. Part (b) maps this architecture to the protocol executed for a `select` query.

Figure 1: Proxy-Based Cedar Implementation

database access is required while disconnected. Cedar's central organizing principle, as stated earlier in Section 1, is that even a stale client replica can be used to reduce data transmission volume. There is no expectation that a tentative result from a client replica will be correct, only that it will be "close" to the authoritative result at the server. Thus, the output of a client replica is never accepted without first checking with the server. This "check-on-use" or detection-based approach to replica control was originally developed in the context of distributed file systems such as AFS-1 [29]. We rejected a "notify-on-change" or avoidance-based approach even though it has been shown to have superior performance in AFS-2 and its successors. There were multiple reasons for this decision. First, it reduces wasteful invalidation traffic to a weakly-connected client in situations where most of the invalidations are due to irrelevant update activity. Second, we were concerned about the performance issues that may arise on a busy database server that has to actively track state on a large number of clients. Finally, it simplifies the implementation at both the client and the server

Cedar is implemented in Java, and runs on Linux as well as Windows. It should also work on any other operating system that supports Java. In the next four sections, we examine specific aspects of Cedar's design. Section 3.1 discusses how Cedar is made transparent to both applications and databases. Section 3.2 describe how Cedar detects commonality across results and constructs compact result summaries. Section 3.3 describes Cedar's support for creating partial replicas. Section 3.4 discusses how stale client replicas can be refreshed.

3.1 Proxy-based Transparency

A key factor influencing our design was the need for Cedar to be completely transparent to both applications and databases. This lowers the barrier for adoption of Cedar, and broadens its applicability. We use a proxy-based design to meet this goal. Our task is simplified by the fact that Cedar is a pure performance enhancement, and not a functionality or usability enhancement.

3.1.1 Application Transparency

Cedar does not require access to application source code. Instead, we leverage the fact that most applications that access databases are written to a standardized API. This compact API (JDBC in our implementation) is a convenient interposition point for new code. Figure 1(a) shows how we interpose Cedar. On the application end, the native database driver is replaced by Cedar's drop-in replacement driver that implements the same API. The driver forwards all API calls to a co-located proxy.

Figure 1(b) shows how these components interact when executing a `select` query. The proxy first executes the query on the client replica and generates a compact CAS description of the result. It then forwards the original query and the CAS description to the database server. Note that while one could combine Cedar's database driver and proxy for performance reasons, separating them allows the proxy to be shared by different applications. While Cedar currently interposes on the JDBC interface, it can also support ODBC-based C, C++, and C# applications by using an ODBC-to-JDBC bridge.

3.1.2 Database Transparency

As Figure 1(b) shows, the client's query and CAS description is received by a corresponding server proxy. The server proxy could either be co-located with the server or placed on a separate machine. The only restriction is that it should be well connected to the server. This proxy re-executes the query on the server and generates a CAS description of the result. It then compares the client's and server's descriptions of the results, eliminates commonality, and only sends back the differences. Note that the database server is totally unaware of the client replica – it is only the server proxy that is aware.

On both the client and the database server, we chose not to incorporate Cedar's functionality directly into the database. Instead, Cedar itself uses the JDBC API to access both the server and client databases. This design choice to allow Cedar to be completely independent of the underlying database has a number of advantages. First, the increased diversity in choice can be very useful when a mobile client lacks sufficient resources to run a heavyweight database server. In such situations, a lightweight or embedded database (such as HSQLDB or SQLite) could be used on the mobile client while a production database such as MySQL or Oracle could be used on the server. Second, it makes a database's physical layout completely transparent to Cedar. Thus, a centralized database in a data center can be easily replicated for scalability reasons without modifying Cedar.

3.1.3 Adaptive Interpositioning

Although a Cedar client is optimized for weakly-connected operation, it may sometimes experience good connectivity. In that situation, its low-bandwidth optimizations may be counter-productive. For example, as shown in Figure 1 (b), the latency added by executing the query twice may exceed the savings from reduced data transmission volume. There may also be situations in which a mobile client is so resource-poor that our approach of using its disk storage and processing power to compensate for weak connectivity is infeasible.

Cedar therefore adapts its behavior to available bandwidth. When the client proxy detects a fast network, it stops handling new queries. The Cedar JDBC driver then transparently switches to direct use of the server proxy. This eliminates the computational overhead of using CAS and the cost of an extra hop through the client proxy. Transactions that are currently in progress through the proxy are completed without disruption. If, at a later point in time, the client proxy detects unfavorable network conditions, it is once again transparently interposed into the query handling path. Bandwidth monitoring is done by the client proxy using packet-pair based IGI/PTR [30]. Our experiments indicate that relatively coarse bandwidth estimation is adequate for triggering proxy state changes.

While we could have also bypassed the server proxy, this would require dynamic substitution of the native JDBC driver for Cedar's JDBC driver. It would be difficult to implement this in a manner that does not disrupt queries in progress. To preserve transparency, we have chosen to forgo this optimization. Fortunately, the measurements shown in Section 5 indicate that server proxy overhead is low.

We are also in the process of implementing adaptation with respect to the staleness of the client replica. A very stale replica is likely to produce tentative results that have little commonality with authoritative results. In that situation, it is better to stop using the client replica until it can be brought in sync with the server. Our implementation approach is to have the client proxy track the savings from eliding commonality. When the savings are consistently below a threshold, the interposition of the client proxy is removed. Queries then go directly to the server proxy, as described above.

3.2 Commonality Detection

3.2.1 Exploiting Structure in Data

Rabin fingerprinting is an excellent tool for detecting commonality across opaque objects [14, 42]. An attractive property of this tool is that it finds commonality even after in-place updates, insertions, and deletions are performed on an object. Unfortunately, the data boundaries found by Rabin fingerprinting rarely aligns with natural boundaries in structured data. This makes it less attractive for database output, which is typically organized by rows and columns. Simple reordering of rows, as might occur from a `SORT BY` clause in a SQL statement, can degrade the ability of Rabin fingerprinting to find commonality.

Cedar therefore uses an approach that has worked better than Rabin fingerprinting for us in the past: we use the end of each row in a database result as a natural chunk boundary [51]. It is important to note that Cedar's use of tabular structure in results only involves shallow interpretation of Java's result set data type. There is no deeper semantic interpretation such as understanding data type values, result schema, or integrity constraints.

3.2.2 Generating Compact CAS Descriptions

As Figure 2 shows, Cedar finds commonality by hashing at two granularities: the entire result and each individual row. For large results, the hash per row can add up to a sizable CAS description. We

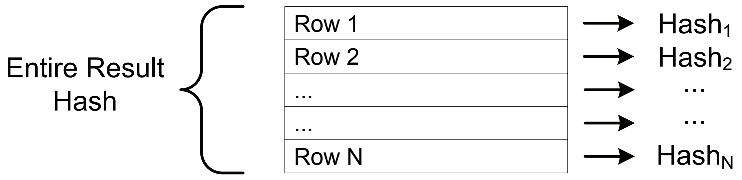


Figure 2: Hashing in Cedar

therefore use a simple but effective approach to reducing the per-row contribution. Our approach recognizes that the entire result hash is sufficient to ensure correctness. Since the sole purpose of a per-row hash is to support comparisons at fine granularity, a subset of the bits in that hash is adequate to serve as a strong hint of similarity. Hence, the CAS description of a result only uses the lower n bits of each per-row hash. When two per-row hashes match, the server assumes that the corresponding row already exists in the tentative result. If this is an erroneous assumption it will be detected in the final step of the comparison process.

After the comparison of tentative and authoritative results is complete, the server sends the client truncated hashes for the common rows found, data for the rows known to be missing on the client, and an *untruncated* hash of the authoritative result. The client reconstructs its version of the authoritative result and verifies that its entire hash matches that sent by the server. In the rare event that there is a mismatch, the authoritative result is fetched verbatim from the server.

A low value of n increases the probability of a collision, and hence many verbatim fetches. On the other hand, a large value of n renders a CAS description less compact. Given a hash of length n bits, the probability p of a collision amongst m other hashes is

$$\begin{aligned} p &= 1 - \Pr\{\text{No Collision}\} \\ &= 1 - \left(\frac{2^n - 1}{2^n}\right)^m \end{aligned}$$

Cedar currently uses a value of $n = 32$ bits. As only result hashes from the *same* query are compared, we expect m to be small. However, even with $m = 10,000$, the probability of seeing a collision and having to refetch verbatim results from the server would only be $p = 2.3 \times 10^{-6}$. It should be noted that we have never encountered a collision during the development and evaluation of Cedar.

3.3 Creating Partial Client Replicas

Some mobile clients may be too resource-poor to support a full database replica. Even when resources are adequate, there may be privacy, security, or regulatory concerns that restrict copying an entire database to an easily-compromised mobile client. These considerations lead to the need for creating partial client replicas. Cedar's challenge is to help create a partial replica that is customized for a mobile user. Fortunately, there are many business and social practices that are likely to result in temporal locality in the query stream at a Cedar client. A partial replica that is tuned to this locality will help Cedar achieve good performance.

```
USERS(nickname, fullname, password, zip)
```

(a) Schema of table named `USERS`

```
<attr table="USERS" predicate="zip >= 15222 AND zip <= 15295"/>
```

(b) Hoard attribute

A hoard attribute is expressed in a notation that is almost identical to the predicate of a `WHERE` clause in a `SELECT` query. The hoard attribute in (b) would cache all rows of the table specified in (a) whose zip codes lie between 15222 and 15295.

Figure 3: Syntax of Hoard Attributes

```
INSERT INTO users VALUES ("jd_nick", "John Doe", "jd_password",
                          15213)
```

(a) Original query from log

```
INSERT INTO users VALUES ("string", "string", "string", number)
```

(b) Abstracted query produced

Figure 4: Query Abstraction by SLM

For example, consider a company that assigns a specific geographic territory or a specific set of customers to each of its salesmen. The query stream generated by a salesman's mobile client is likely to be confined to his unique territory or unique set of customers. Although the company's customer database may be enormous, only a small subset of it is likely to be relevant to that salesman. Similar reasoning applies to an insurance adjuster who is visiting the homes of customers affected by a natural disaster. There is typically a step in the workflow of claim handling that assigns a specific adjuster to each claim. These assignments are typically made at the granularity of many hours, possibly a whole day. For the entire duration of that period, an adjuster is likely to only generate queries that pertain to his assignments.

Our solution is inspired by Coda's use of *hoarding* to support disconnected operation [32]. Cedar faces a simpler problem than Coda because its hoarding does not have to be perfect. If a partial replica cannot produce the correct response to a query, the only consequence in Cedar is bad performance. No disruptive error handling is needed relative to availability or consistency.

3.3.1 Hoard Granularity

Previous implementations of hoarding have typically operated at the granularity of individual objects, such as files or mail messages. In contrast, Cedar hoards data at the granularity of tables and table fragments. Cedar's approach is based on the long-established concepts of *horizontal fragmentation* [15] and *vertical fragmentation* [39], as shown in Figure 5. Horizontal partitioning preserves the database schema. It is likely to be most useful in situations that exploit temporal locality, such as the sales and insurance examples mentioned earlier. Hoarding in Cedar is done through horizontal fragmentation. If a query references a table that is not hoarded on a mobile client, Cedar forwards that query directly to the database server.

Original Table

ID	Name	Address	Zip	Email
1	John Doe	412 Avenue	15213	jd2@eg.com
2	Richard Roe	396 Road	15217	rr@eg.com
3	Mary Major	821 Lane	15232	mm@eg.com
4	John Stiles	701 Street	94105	js@eg.com
5	Jane Doe	212 Way	94112	jd@eg.com

Horizonal Partitioning

ID	Name	Address	Zip	Email
1	John Doe	412 Avenue	15213	jd2@eg.com
2	Richard Roe	396 Road	15217	rr@eg.com
3	Mary Major	821 Lane	15232	mm@eg.com

ID	Name	Address	Zip	Email
4	John Stiles	701 Street	94105	js@eg.com
5	Jane Doe	212 Way	94112	jd@eg.com

Vertical Partitioning

ID	Name	Address
1	John Doe	412 Avenue
2	Richard Roe	396 Road
3	Mary Major	821 Lane
4	John Stiles	701 Street
5	Jane Doe	212 Way

ID	Zip	Email
1	15213	jd2@eg.com
2	15217	rr@eg.com
3	15232	mm@eg.com
4	94105	js@eg.com
5	94112	jd@eg.com

Figure 5: Horizontal and Vertical Partitioning

3.3.2 Database Hoard Profiles

Hoarding at a Cedar client is controlled by a *database hoard profile* that expresses hoard intentions within the framework of a database schema. A database hoard profile is an XML file that contains a list of weighted *hoard attributes.* Each hoard attribute specifies a single database relation and a predicate. Cedar uses these predicates to horizontally partition a relation. A database hoard profile may be manually created by a user. However, it is much more likely to be created by a database administrator or by using the tool described in Section 3.3.3.

Figure 3 illustrates the syntax of hoard profiles. Each hoard attribute is associated with a weight that indicates its importance. Cedar uses this weight to prioritize what should be hoarded on a mobile client that has limited storage. As tentative results are always verified with the server, supporting external referential constraints, such as foreign keys, is not required for correctness. However, if needed, extending hoard profiles to use referential cache constraints [4] should be simple.

3.3.3 Tools for Hoarding

To specify hoard attributes, a user needs to be aware of the SQL queries that she is likely to generate while weakly connected. It is often the case that a user does not directly specify SQL queries, but indirectly generates them through an application. To help the user in these situations, we have developed a tool called *SQL Log Miner (SLM)*. This tool analyzes query logs to aid in the creation of database hoard profiles. It first abstracts queries into templates by removing unique user input, as illustrated in Figure 4. It then analyzes the abstracted log to determine unique queries and outputs them in frequency sorted order. SLM is also able to use a database's `EXPLAIN` feature to display queries that generate the largest results.

Application	Total Queries	Unique Queries	Unique %
`AUCTION`	115,760	41	0.04 %
`BBOARD`	131,062	59	0.05 %
datapository.net	9,395,117	278	0.003 %

Table 2: Unique Queries Within Workloads

We used SLM to analyze database logs from a number of applications traces including an auction benchmark [6], a bulletin board benchmark [6], and `datapository.net`, a web-enabled database used for network trace analysis by the academic community. As Table 2 shows, the number of unique queries was very small relative to the large number of queries. The data in Table 2 suggests that it may be tractable to extend SLM to automate hoard profile creation. Contextual information [43] and data mining [56] may be additional sources of input for this automation task.

3.4 Refreshing Stale Client Replicas

Cedar offers two mechanisms for bringing a client replica in sync with the database server. If a client has excellent connectivity, a new replica can be created or a stale replica updated by simply restoring a database dump created using the client's hoard profile. Although this is bandwidth-intensive, it tends to be faster and is our preferred approach.

To handle extended periods of weak connectivity, Cedar also provides a log-based refresh mechanism. The database server continuously maintains a timestamped update log. Since logging is a lightweight operation, it typically has less than a 1% impact on performance [38]. When a client detects available bandwidth, it can obtain the log from the server and apply all the updates since the last refresh. Bandwidth detection mechanisms can ensure that the log fetch does not impact foreground workloads. The server allocates a finite amount of storage for its log, and recycles this storage as needed. It is the responsibility of clients to obtain log entries before they are recycled. Once a log entry is recycled, any client that needs it has to restore its entire replica using the method described in the previous paragraph.

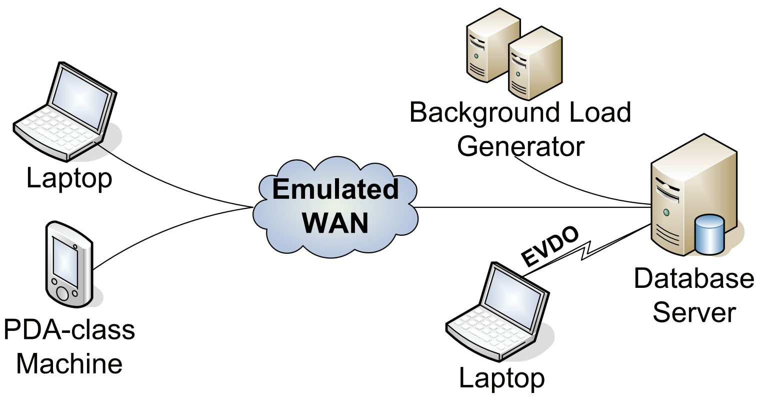


Figure 6: Experimental Setup

4 Evaluation Approach and Setup

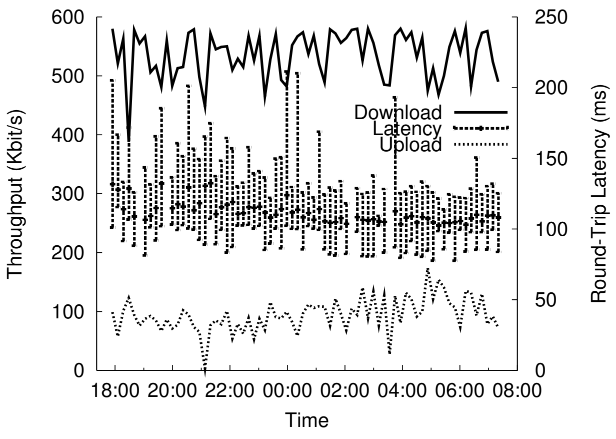
How much of a performance win can Cedar provide? The answer clearly depends on the workload, the network quality, and on the commonality between the client replica and the server. For resource-poor clients, it can also depend on the computational power of the client. To obtain a quantitative understanding of this relationship, we have conducted experiments using both micro- and macro-benchmarks. This section presents our experimental setup. Our benchmarks and results follow in Sections 5 and 6.

The experimental setup used to evaluate Cedar is shown in Figure 6. The database server and the background load generator were 3.2 GHz Pentium 4s (with Hyper-Threading) with 4 and 2 GB of RAM respectively. The laptop was a Thinkpad T41p with a 1.7 GHz Pentium M processor and 1 GB of RAM. The PDA-class machine (henceforth referred to as just PDA) was a 1997-era Thinkpad 560X with a 233 MHz Pentium MMX processor and 64 MB of RAM.

All the machines ran the Fedora Core 5 Linux distribution with the 2.6.18-1 SMP kernel. We used Sun's Java 1.5 as Cedar's runtime environment. The database server and the client replica used the open source MySQL database. The server proxy was located on the same machine as the database server.

We use the approach described in Section 3.2.1 to detect and elide commonality between results. As CAS has been shown to be better than compression [37], we do not consider the latter in our evaluation.

As shown in Figure 6, we used both real and emulated networks to evaluate Cedar's performance. For the real Wireless WAN, we used a Novatel Wireless (Merlin S720) PCMCIA card to connect the laptop directly to the database server over a Sprint EVDO (Rev. A) network link. While the theoretical throughput for this network is 3,100 Kbit/s down and 1,800 Kbit/s up, Sprint advertises 450–800 Kbit/s down and 300–400 Kbit/s up. To verify this, we used Iperf [31] and *ping* to measure the bandwidth and round trip latency between the EVDO-enabled laptop and the database server. The network was probed every 10 minutes over an $\sim$13 hour period. The throughput was averaged over a 20 second transfer and the round-trip latency over 25 pings. The measurements, presented in Figure 7, showed an average download and upload bandwidth of 535 and 94 Kbit/s



The error bars for the latency measurements indicate the maximum and minimum observed values. Four outliers (5X–30X of the mean) were removed from the ping data.

Figure 7: EVDO Network Throughput and Latency

and an average round-trip latency of 112 ms.

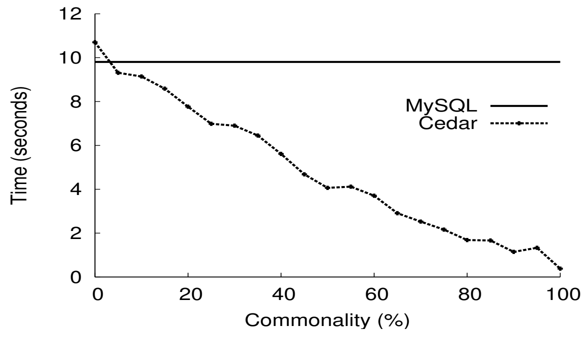
Based on these measurements and the theoretical ratings of other wireless technologies shown in Table 1, we decided to evaluate Cedar with emulated networks of 100 Kbit/s (representative of CDMA 1xRTT), 500 Kbit/s and 1 Mbit/s (representative of EVDO), and 2 Mbit/s (representative of WiMax). On each network, we experimented with round-trip times of 33, 66, and 100 ms. The emulated WAN was implemented as a NetEm bridge [27]. As our focus was on the performance of the mobile client, there were no constraints on the network link between the database and the background load generator.

5 Microbenchmarks

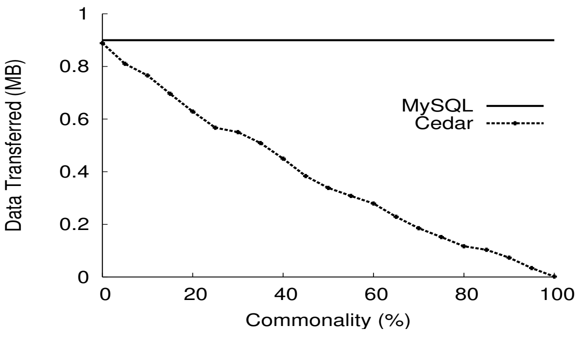
We used sensitivity analysis to explore how Cedar performs with different hit rates on the client replica. We executed a number of single-query microbenchmarks where we varied both the amount of data selected by the query and the fraction of the result data that was available on the mobile client. The queries in each microbenchmark fetched approximately 0.5, 1, and 5 MB of data. For each of the microbenchmarks, the fraction of data present on the mobile client was varied between 0% and 100%. For example, in the 60% case, the client replica was explicitly populated with 60% of the data that would be selected by the micro-benchmark's query. The two extreme hit-ratios of 100% and 0% give an indication of best-case performance and the system overhead. To provide a baseline for comparison, we also ran the benchmarks with MySQL's native JDBC driver.

5.1 Results

In the interests of brevity, we only present details of a subset of the microbenchmark results and summarize the rest. For the microbenchmark that fetched approximately 1 MB of data, Figure 8 presents the query completion time and the amount of data downloaded by the mobile client. The



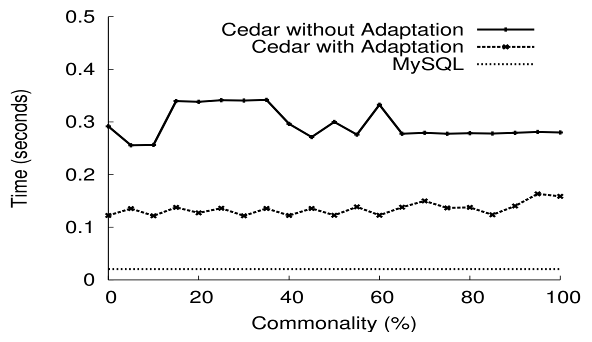
(a) Query Latency



(b) Data Transferred

The microbenchmark was executed on the laptop. Results are the mean of three trials. The maximum standard deviation for data transferred was 2% of the corresponding mean. There was no significant standard deviation for the query execution time.

Figure 8: 1 MB Microbenchmark: 1 Mbit/s and 100ms Network



The microbenchmark was executed on the laptop. Results are the mean of three trials and had no significant standard deviation.

Figure 9: 1 MB Microbenchmark Query Latency: Gigabit

results show that Cedar's performance is almost a linear function of the amount of data found on the client replica. Cedar's overhead is noticeable only when the degree of commonality falls below 5–10%. Irrespective of whether the laptop or PDA was used, the same trends were observed for all the other microbenchmarks and network configurations.

Apart from the networks described in Section 4, we also ran the microbenchmarks over a Gigabit Ethernet link. While Cedar is not geared towards extremely fast networks, this allowed us to evaluate Cedar's adaptability in a high bandwidth setting. As shown in Figure 9, Cedar, without adaptation, can be an order of magnitude slower than MySQL. Our analysis showed that the most significant cause for this slowdown was the extra software hops through Cedar's client and server proxy. A smaller, but nevertheless noticeable, fraction of the overhead was the computation cost of CAS. As described in Section 3.1.3, whenever the client detects a fast network, it will bypass the local proxy and switch to directly contacting the server-side proxy. This adaptation gets Cedar's overhead down to within 5X of the native performance. While still relatively high, the absolute

Interaction Type	Percentage
Create New Customer	1%
Change Payment Method	5%
Create Order	50%
Order Status	5%
View New Products	7%
View Product Detail	30%
Change Item	2%

Table 3: Default Mix of Client Interactions

difference is within 0.1 seconds of native performance. Tuning our unoptimized prototype should further improve Cedar's performance.

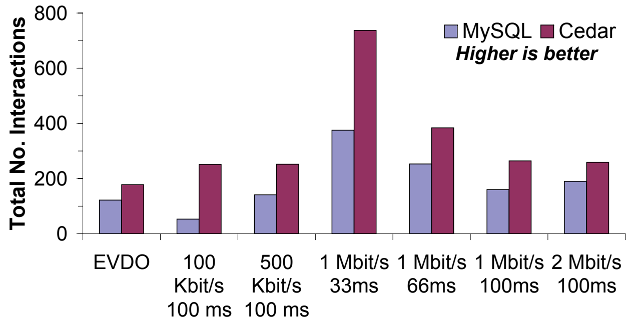
6 The MobileSales Benchmark

We are not aware of any widely used benchmark for evaluating mobile access to databases. We have therefore created MobileSales, a new benchmark based on the recently announced TPC-App [54] benchmark from the Transaction Processing Performance Council. While TPC-App is targeted towards an application server environment, we believe that a modified version of this benchmark is also applicable to the mobile scenario.

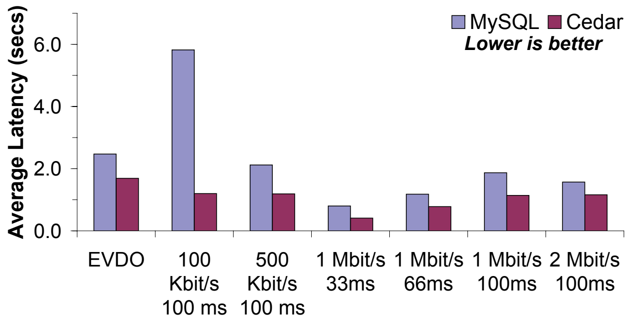
The TPC-App benchmark consists of an online distributor that allows clients to interact with a sales system. The workload consists of a set of *interactions*: clients can add new customers, create new orders, change payment types, check on the status of previous orders, view new products the distributor might have recently added, look at detailed descriptions of products, and make changes to the product catalog. Each client's test run is defined by a time period that specifies the length of benchmark execution and a *mix* that specifies the probability of each interaction type. The default mix is shown in Table 3. There is no think time between interactions. The benchmark dataset can be scaled with respect to the number of customers. The dataset size is defined by $(0.8 \times S + 2006)$ MB where S is the scale factor.

While the focus of TPC-App is to test application server performance, we believe that the benchmark's workload is representative of a number of mobile use scenarios including traveling salesmen, insurance adjusters, and customer relation management (CRM) applications. MobileSales therefore retains the same workload (dataset, queries, and interaction types) but, unlike TPC-App, allows mobile clients to directly access the database instead of being mediated through the application server. This is not uncommon today and is recommended for enterprise applications on mobile clients [58].

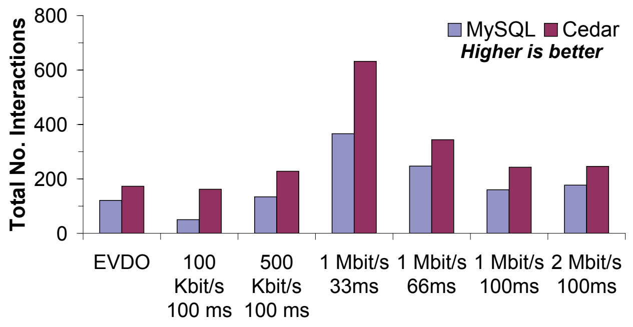
The performance of an individual mobile client is measured in terms of throughput (total number of interactions) and latency (average interaction completion time). MobileSales can also execute clients on separate load generator machines. Apart from modeling concurrent database access, updates made by these load clients ensure that the client replica diverges from the server as the benchmark progresses.



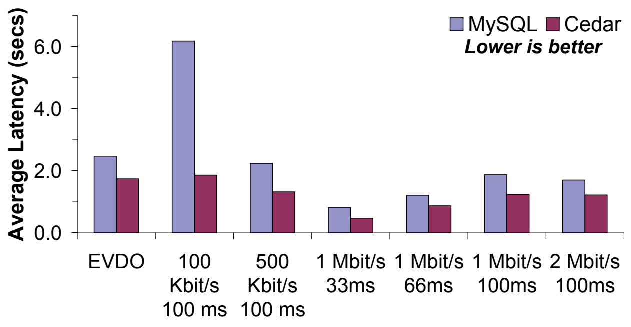
(a) Throughput: Unloaded Server



(b) Latency: Unloaded Server



(c) Throughput: Server Load = 50



(d) Latency: Server Load = 50

All results are the mean of three trials. The maximum standard deviation for throughput and latency was 6% and 7% respectively of the corresponding mean.

Figure 10: Mobile Sales - Laptop with Full Hoard Profile

6.1 Benchmark Setup

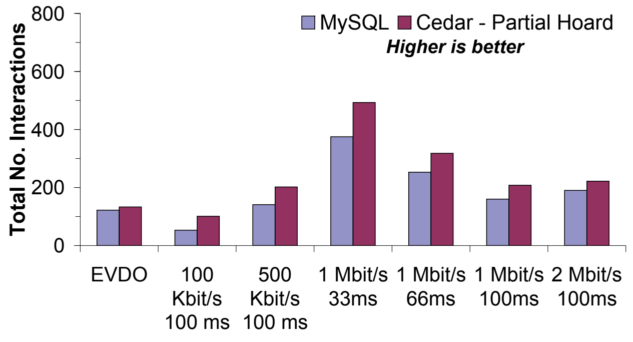
For our experiments, we set the scale factor S to 50. This generates a dataset large enough for simultaneous access by 50 clients. While the raw data size is 2 GB, it occupies 6.1 GB on disk due the addition of indexes and other MySQL metadata.

Each test run executed for five minutes using the default mix of client interactions shown in Table 3. This mix has a 42:58 read:write ratio. The high write ratio biases the results against Cedar as updates are not optimized. A higher read ratio would only increase Cedar's benefits.

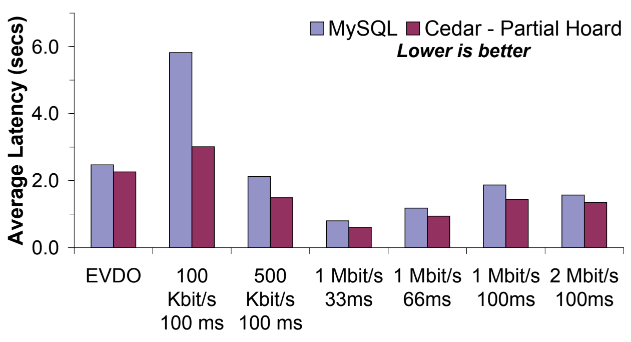
During different benchmark runs, we set the number of load clients to either 0, 10, 30, or 50. In the interests of space, we only present the results for the unloaded server (0 load clients) and 50 clients. The results with 10 and 30 clients fall in between the 0 and 50 clients cases. The baseline for comparison is direct database access via MySQL's native JDBC driver. Relative to the baseline, improvement is defined as

$$Improvement = \frac{Result_{Cedar} - Result_{Native}}{Result_{Native}}$$

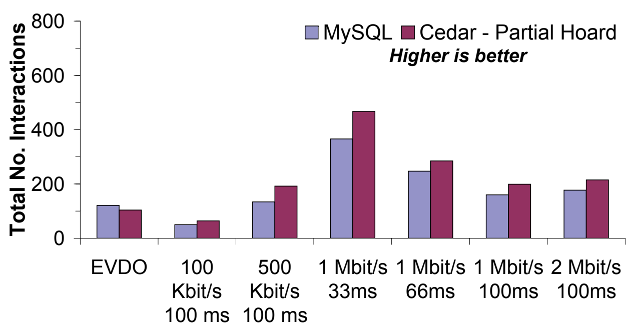
We also evaluated the performance of mobile clients with two different hoard profiles. The first profile, named Full Hoard, selected the entire database for replication. The second profile, named Partial Hoard, only selected half of the product catalog (a predominantly read-only portion of the



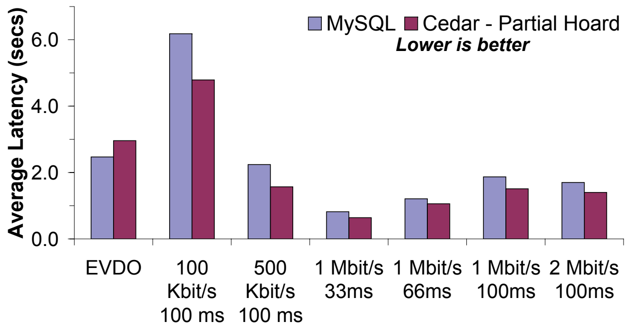
(a) Throughput: Unloaded Server



(b) Latency: Unloaded Server



Throughput: Server Load = 50



Latency: Server Load = 50

All results are the mean of three trials. The maximum standard deviation for both throughput and latency was 8% of the corresponding mean.

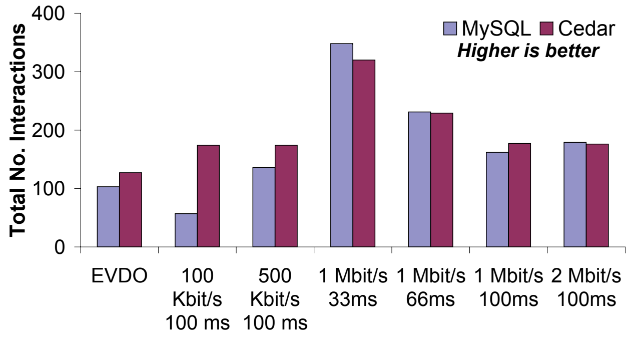
Figure 11: Mobile Sales - Laptop with Partial Hoard Profile

database) and did not include customer or order information.

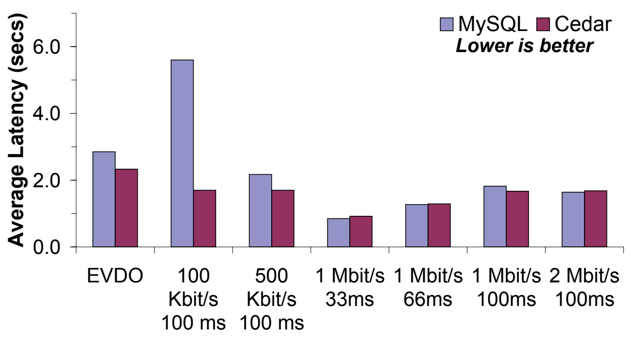
6.2 Laptop: Full Hoard Results

Figure 10 presents the results of the MobileSales benchmark with the Full hoard profile. We first examine the unloaded server case where the laptop is the only client accessing the database. The results, presented in Figures 10 (a) and (b), show that the throughput improvements due to Cedar range from 36% in the high bandwidth case of 2 Mbit/s to as much as 374% in the low bandwidth case of 100 Kbit/s. The average interaction latency shows similar improvements, ranging from a 26% reduction at 2 Mbit/s to a 79% reduction at 100 Kbit/s. The results from the emulated networks match the real EVDO network where Cedar shows a 46% improvement in throughput and a 30% reduction in the average interaction latency. As latency on the 1 Mbit/s network increases, throughput drops for both MySQL and Cedar. This occurs because neither system can fill the bandwidth-delay product.

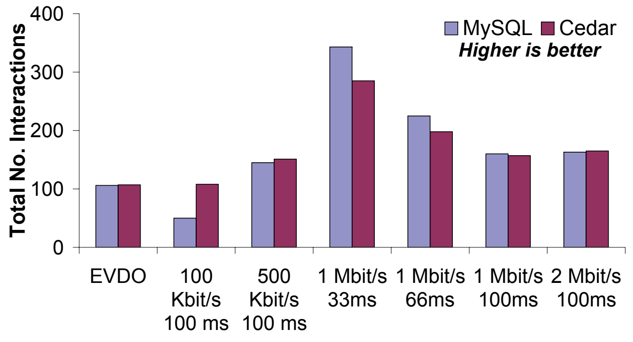
Figures 10 (c) and (d) show that even when the number of load clients is increased to 50, Cedar retains its performance advantage. MySQL's performance shows very little change when compared to an unloaded server. As the database is not the bottleneck, the impact on both MySQL's through-



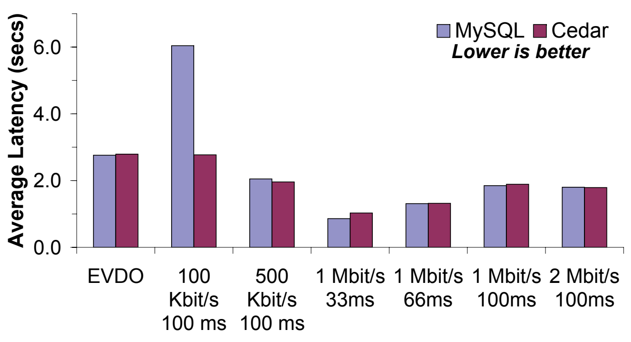
(a) Throughput: Unloaded Server



(b) Latency: Unloaded Server



(c) Throughput: Server Load = 50



(d) Latency: Server Load = 50

All results are the mean of three trials. The maximum standard deviation for throughput and latency was 11% and 15% respectively of the corresponding mean.

Figure 12: Mobile Sales - PDA with Full Hoard Profile

put and average interaction latency is less than 8%. Cedar, when compared to the unloaded server, does show a more noticeable drop. As updates made by the load clients increase the divergence between the client replica and the server, Cedar has to fetch more data over the low bandwidth network. Compared to the unloaded server, Cedar experiences a 3% to 35% drop in throughput and a 3% to 55% increase in latency. However, it still performs significantly better than MySQL in all experimental configurations. The throughput improvements range from 39% at 2 Mbit/s to 224% at 100 Kbit/s and latency reductions range from 28% at 2 Mbit/s to 70% at 100 Kbit/s.

6.3 Laptop: Partial Hoard Results

Figure 11 presents the results of the MobileSales benchmark with the Partial Hoard profile. As the hoard profile has no impact on MySQL, its results are unchanged from Section 6.2. While the performance gains due to Cedar drop when compared to the Full hoard profile, they are still substantial. For an unloaded server, as seen in Figures 11 (a) and (b), Cedar can deliver a throughput improvement ranging from 9% at EVDO to 91% at 100 Kbit/s and an average interaction latency reduction ranging from 9% at EVDO to 48% at 100 Kbit/s. Even for the faster 1 and 2 Mbit/s networks, Cedar's throughput and latency improvements are in the range of 17–31% and 14–24%.

Figures 11 (c) and (d) show that as Cedar's throughput drops when compared to an unloaded server, the improvement relative to MySQL now ranges from a throughput increase of 15% at 1 Mbit/s with 33ms to 43% at 500 Kbit/s. The corresponding reduction in average interaction latency ranges from 12% at 1 Mbit/s with 66ms to 30% at 500 Kbit/s. The only exception is the EVDO network where Cedar's throughput is 14% lower than MySQL. We suspect that this was a temporary network phenomena as these EVDO results exhibited the greatest variance in the experiments.

6.4 PDA: Full Hoard Results

Figure 12 presents the results from running MobileSales on the PDA with the Full hoard profile. We see that, even with an extremely resource-limited client, Cedar can still deliver a significant performance improvement. With the EVDO network and an unloaded server, we see a 23% improvement in throughput and 18% reduction in latency. For the 100 Kbit/s network, we see a 205% improvement in throughput and 70% reduction in latency for an unloaded server and a 116% improvement in throughput and 54% reduction in latency with 50 load clients.

The gains from using Cedar tail off at the higher bandwidths with equivalent performance in most cases. However, Cedar actually performed slightly worse than MySQL on a 1 Mbit/s with 33ms network. Comparing it to MySQL's throughput, we see a drop of 8% with an unloaded database server and a drop of 17% with 50 load clients. A similar drop is seen with 50 load clients on a 1 Mbit/s with 66ms network. This performance drop arose because the 233 MHz CPU became a bottleneck when large amounts of data needed to be hashed. This is exactly the scenario where the adaptation technique proposed in Section 3.1.3 would be useful. Tuning Cedar for PDAs should further decrease this overhead.

6.5 PDA: Partial Hoard Results

Switching the PDA from the Full hoard profile to the Partial hoard profile showed the same behavior as with the laptop. We therefore omit a detailed description of these results in the interests of space. Overall, Cedar delivered equivalent or better performance that MySQL in almost all of the network configurations. The only exceptions were the two 1 Mbit/s network configurations highlighted in Section 6.4. In these cases, relative to MySQL, Cedar's worst case overhead for throughput and latency was 11% and 12% respectively.

6.6 Summary

Our results from the MobileSales benchmark demonstrate that Cedar can significantly improve performance in low-bandwidth conditions. We view these results as especially encouraging as they were achieved without any compromise in consistency and without any modifications to the application or database. Also, as described in Section 6.1, these results arise from a write-intensive workload that is biased against Cedar. We predict that Cedar would perform even better for workloads with a greater percentage of reads. Further, while Cedar is primarily targeted towards laptop-class clients, our results indicate that even resource-limited PDA-class machines can benefit.

7 Related Work

Our work has been strongly influenced by previous work that also aims at improving performance over WANs. In this section, we describe these systems and highlight Cedar's similarities and differences. In particular, we discuss systems that relax consistency for improved performance, mobile database systems, and other systems that use hash-based techniques.

7.1 Relaxing Consistency

While the benefits of caching data in mobile systems has long been known [17], most systems weaken traditional consistency semantics for performance reasons. Weakly-Consistent replication in Bayou [50] was proposed for collaborative systems. Alonso et al. [2] proposed *quasi-copies* for database systems to improve performance by relaxing consistency when clients participate in caching data. The Mariposa [47] distributed data manager showed how consistent, although potentially stale, views can be seen across a network of machines. Gray et al. [25] proposed a two-tier replication scheme that allows for tentative update transactions on the mobile client that are later committed to the master copy.

Like Cedar, a number of systems have advocated middle-tier database caching where parts of the database are replicated at the edge for web-based applications [3, 4, 5, 33]. These systems, based on an avoidance-based approach, require tight integration with the database to ensure timely propagation of updates and are targeted towards workloads that do not require strict consistency.

Gao et al. [24] propose using a distributed object replication architecture where the data store's consistency requirements are adapted on a per-application basis. These solutions require substantial developer resources and detailed understanding of the application being modified. While systems that attempt to automate the partitioning and replication of an application's database exist [48], they do not provide full transaction semantics.

An obvious disadvantage of these systems is that they provide a different consistency or transactional model than what developers and users have grown to expect. Tentative transactions increase the probability of conflicts and require additional application complexity or user interaction for conflict resolution. In contrast, Cedar's focus is on maintaining the transactional and consistency semantics provided by the underlying database. While this design decision prevents disconnected operation, we believe that this is an acceptable tradeoff for an important class of real-world usage scenarios.

7.2 Mobile Database Systems

Barbará and Imielinski [9] suggest informing clients of changes made to a centralized database by broadcasting Invalidation Reports (IRs) at periodic intervals. However, as IRs are a push-based system, they are only effective if a large number of clients are interested in the same data. While hybrid push-pull systems have been proposed [1], in the absence of locality, they still degenerate into a pull-based behavior. IRs also add significant latency to queries as each mobile client has to wait for the periodic IR broadcast before it can verify data freshness. For a complete survey of previous work on IRs and the mobile databases described above in Section 7.1, we refer the reader to Barbará [8].

Some systems [8, 57] allow clients to obtain an exclusive copy of the section of the database it is accessing. This can significantly degrade performance for other clients when mobile clients are weakly connected. By recognizing that the local database replica could be stale, Cedar instead ensures that there is no strong dependency between the main database server and the mobile client.

Cedar's hoard profiles also bear some similarity to clustering attributes [7] and client profiles [13]. However, while all three are used to express preferences for database content, they have very different functions. Clustering attributes define a database server's storage layout for improved access by mobile clients while client profiles are used to indicate freshness and latency requirements for systems that relax consistency.

7.3 Hash-based Systems

Successful use of CAS span a wide range of storage systems. Examples include peer-to-peer backup of personal computing files [20], storage-efficient archiving of data [11, 41], remote file synchronization [55], DHT-based storage systems [21, 22], and file similarity detection [35].

Spring and Wetherall [49] apply similar principles at the network level. Using synchronized caches at both ends of a network link, duplicated data is replaced by smaller tokens for transmission and then restored at the remote end. This and other hash-based systems such as the CASPER [52] and LBFS [37] file systems, and Layer-2 bandwidth optimizers such as Riverbed and Peribit use Rabin fingerprinting [42] to discover spans of commonality in data. This approach is especially useful when data items are modified *in-place* through insertions, deletions, and updates. However, as shown previously [51], using structural information found in query results can lead to a significant performance improvement than Rabin fingerprinting-based approaches.

8 Future Work and Conclusion

Cedar has shown the benefits of using client replicas. However, hoarding data on a mobile client can have privacy, security, and regulatory concerns as the loss or theft of a mobile device can expose confidential data. While vertical fragmentation, described in Section 3.3.1, can be used to elide sensitive database fields, it can also lead to a performance loss if queries frequently reference those fields. Another possible alternative for data protection is to encrypt sensitive data. Encryption could be implemented in a number of ways, including at the storage layer [44], within the client replica [26], or even within the client proxy [19]. We are currently evaluating these choices in order to minimize encryption's impact on resource-limited clients.

We have not considered Cedar's impact on power consumption. While reducing network transmission has been shown to save power, there is a tradeoff as Cedar uses computational cycles to achieve the reduction. A careful investigation is needed to determine how power should be factored into Cedar's adaptation mechanisms.

In summary, this paper presents the design and implementation of Cedar, a system that enables mobile database access with good performance while preserving consistency. Cedar leverages a mobile client's storage and computational resources to compensate for weak connectivity. Our results demonstrate that Cedar has low overhead and can significantly improve performance for its intended usage environment.

Acknowledgments

We are grateful to Dave Andersen and Amit Manjhi for providing database server logs, Stratos Papadomanolakis and Amol Deshpande for their feedback on the MobileSales application, and Jan Harkes for their help in resurrecting the PDA-class machines. We are also thankful to include Dave Andersen, Jason Ganetsky, Adam Goode, Andres Lagar-Cavilla, Stratos Papadomanolakis, Joshua Rice, Ajay Surie, and Eno Thereska for their feedback.

References

[1] Swarup Acharya, Michael Franklin, and Stanley Zdonik. Balancing push and pull for data broadcast. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 183–194, May 1997.

[2] Rafael Alonso, Daniel Barbara, and Hector Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transaction on Database Systems*, 15(3):359–384, 1990.

[3] Mehmet Altinel, Qiong Luo, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh, Bruce G. Lindsay, Honguk Woo, and Larry Brown. Dbcache: Database caching for web application servers. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 612–612, 2002.

[4] Mehmet Altinel, Christof Bornhövd, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh, and Berthold Reinwald. Cache tables: Paving the way for an adaptive database cache. In *Proceedings of 29th International Conference on Very Large Data Bases*, pages 718–729, Berlin, Germany, 2003.

[5] Khalil Amiri, Sanghyun Park, Renu Tewari, and Sriram Padmanabhan. Dbproxy: A dynamic data cache for web applications. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, March 2003.

[6] Cristiana Amza, Emmanuel Cecchet, Anupam Chanda, Alan Cox, Sameh Elnikety, Romer Gil, Julie Marguerite, Karthick Rajamani, and Willy Zwaenepoe. Specification and implementation of dynamic web site benchmarks. In *Proceedings of the Fifth Annual IEEE International Workshop on Workload Characterization (WWC-5)*, pages 3–13, Austin, TX, November 2002.

[7] B. R. Badrinath and Shirish H. Phatak. On clustering in database servers for supporting mobile clients. *Cluster Computing*, 1(2):149–159, 1998.

[8] Daniel Barbará. Mobile computing and databases - a survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):108–117, 1999.

[9] Daniel Barbará and Tomasz Imielinski. Sleepers and workaholics: Caching strategies in mobile environments. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 1994.

[10] J. Black. Compare-by-hash: A reasoned analysis. In *Proc. 2006 USENIX Annual Technical Conference*, pages 85–90, Boston, MA, May 2006.

[11] William J. Bolosky, Scott Corbin, David Goebel, , and John R. Douceur. Single instance storage in windows 2000. In *Proceedings of the 4th USENIX Windows Systems Symposium*, pages 13–24, Seattle, WA, August 2000.

[12] Eric A. Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, 2001.

[13] Laura Bright and Louiqa Raschid. Using latency-recency profiles for data delivery on the web. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 550–561, Hong Kong, China, August 2002.

[14] Andrei Broder, Steven Glassman, Mark Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *Proceedings of the 6th International WWW Conference*, pages 1157–1166, Santa Clara, CA, April 1997.

[15] Stefano Ceri, Mauro Negri, and Giuseppe Pelagatti. Horizontal data partitioning in database design. In *SIGMOD '82: Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data*, pages 128–136, June 1982.

[16] Rajiv Chakravorty, Suman Banerjee, Pablo Rodriguez, Julian Chesterfield, and Ian Pratt. Performance optimizations for wireless wide-area networks: Comparative study and experimental evaluation. In *MobiCom '04: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, pages 159–173, 2004.

[17] Boris Y. Chan, Antonio Si, and Hong V. Leong. A framework for cache management for mobile databases: Design and evaluation. *Distributed and Parallel Databases*, 10(1):23–57, 2001.

[18] Mun Choon Chan and Ramachandran Ramjee. Tcp/ip performance over 3g wireless links with rate and delay variation. *Wireless Networks*, 11(1-2):81–97, 2005.

[19] Mark D. Corner and Brian D. Noble. Protecting applications with transient authentication. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 57–70, 2003.

[20] Landon P. Cox, Christopher D. Murray, and Brian D. Noble. Pastiche: Making backup cheap and easy. In *OSDI: Symposium on Operating Systems Design and Implementation*, 2002.

[21] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[22] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, pages 75–80, Schloss Elmau, Germany, May 2001.

[23] EMCCentera03. *EMC Centera Content Addressed Storage System.* EMC Corporation, 2003. http://www.emc.com/.

[24] Lei Gao, Mike Dahlin, Amol Nayate, Jiandan Zheng, and Arun Iyengar. Application specific data replication for edge services. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 449–460, 2003.

[25] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 173–182, June 1996.

[26] Hakan Hacigumus, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 216–227, 2002.

[27] Stephen Hemminger. Netem - emulating real networks in the lab. In *Proceedings of the 2005 Linux Conference Australia*, Canberra, Australia, April 2005.

[28] Val Henson. An analysis of compare-by-hash. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, pages 13–18, May 2003.

[29] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[30] Ningning Hu and Peter Steenkise. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications (J-SAC)*, 21(6): 879–894, August 2003.

[31] Iperf. The tcp/udp bandwidth measurement tool. `http://dast.nlanr.net/Projects/Iperf/`.

[32] James J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computing Systems*, 10(1):3–25, 1992.

[33] Per-Åke Larson, Jonathan Goldstein, and Jingren Zhou. Transparent mid-tier database caching in sql server. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 661–661, 2003.

[34] George Lawton. What lies ahead for cellular technology? *IEEE Computer*, 38(6):14–17, 2005.

[35] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, San Fransisco, CA, 17–21 1994.

[36] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc., 2001.

[37] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Chateau Lake Louise, Banff, Canada, October 2001.

[38] *MySQL 5.0 Reference Manual*. MySQL AB, October 2006.

[39] Shamkant B. Navathe, Stefano Ceri, Gio Wiederhold, and Jinglie Dou. Vertical partitioning algorithms for database design. *ACM Transactions on Database Systems*, 9(4):680–710, 1984.

[40] Timo Ojala, Jani Korhonen, Tiia Sutinen, Pekka Parhi, and Lauri Aalto. Mobile kärpät: A case study in wireless personal area networking. In *MUM '04: Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia*, pages 149–156, 2004.

[41] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, 2002.

[42] Michael Rabin. Fingerprinting by random polynomials. In *Harvard University Center for Research in Computing Technology Technical Report TR-15-81*, 1981.

[43] Murali Rangan, Ed Swierk, and Douglas B. Terry. Contextual replication for mobile users. In *Proceedings of the International Conference on Mobile Business (ICMB'05)*, pages 457–463, Synday, Australia, July 2005.

[44] Erik Riedel, Mahesh Kallahalla, and Ram Swaminathan. A framework for evaluating storage system security. In *Proceedings of the FAST '02 Conference on File and Storage Technologies*, pages 15–30, Monterey, CA, January 2002.

[45] Secure Hash Standard (SHS). Technical Report FIPS PUB 180-1, NIST, 1995.

[46] Secure Hash Standard (SHS). Technical Report FIPS PUB 180-2, NIST, August 2002.

[47] Jeff Sidell, Paul M. Aoki, Adam Sah, Carl Staelin, Michael Stonebraker, and Andrew Yu. Data replication in mariposa. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 485–494, New Orleans, LA, February 1996.

[48] Swaminathan Sivasubramanian, Gustavo Alonso, Guillaume Pierre, and Maarten van Steen. Globedb: Autonomic data replication for web applications. In *WWW '05: Proceedings of the 14th International World-Wide Web conference*, May 2005.

[49] Neil T. Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of ACM SIGCOMM*, August 2000.

[50] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *SOSP '95: Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 172–182, December 1995.

[51] Niraj Tolia and M. Satyanarayanan. No-compromise caching of dynamic content from relational databases. Technical Report CMU-CS-06-146, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 2006.

[52] Niraj Tolia, Michael Kozuch, Mahadev Satyanarayanan, Brad Karp, Adrian Perrig, and Thomas Bressoud. Opportunistic use of content addressable storage for distributed file systems. In *Proceedings of the 2003 USENIX Annual Technical Conference*, pages 127–140, San Antonio, TX, June 2003.

[53] Niraj Tolia, Jan Harkes, Michael Kozuch, and Mahadev Satyanarayanan. Integrating portable and distributed storage. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, San Francisco, CA, March 31 - April 2, 2004.

[54] *TPC Benchmark App (Application Server): Specification.* Transaction Processing Performance Council, San Francisco, CA, 1.1.1 edition, August 2005.

[55] A. Tridgell and P. Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, Department of Computer Science, The Australian National University, Canberra, Australia, 1996.

[56] Qingsong Yao, Aijun An, and Xiangji Huang. Finding and analyzing database user sessions. In *10th International Conference Database Systems for Advanced Applications*, pages 851–862, Beijing, China, April 2005.

[57] Markos Zaharioudakis, Michael J. Carey, and Michael J. Franklin. Adaptive, fine-grained sharing in a client-server oodbms: a callback-based approach. *ACM Transactions onn Database Systems*, 22(4):570–627, 1997.

[58] Bruce Zenel and Andrew Toy. Enterprise-grade wireless. *ACM Queue*, 3(4):30–37, 2005.