Not Your Father's Big Data

Carl Nuessle
University at Buffalo
carlnues@buffalo.edu

Oliver Kennedy
University at Buffalo
okennedy@buffalo.edu

Lukasz Ziarek
University at Buffalo
lziarek@buffalo.edu

ABSTRACT

Embedded database libraries provide developers with a common and convenient data persistence layer. They have spread to many systems, including interactive devices like smartphones, appearing in all major mobile systems. Their performance affects the response times and resource consumption of millions of phone apps and billions of phone users. It is thus critical that we better understand how they work, so they can be used more efficiently, and so developers can make faster libraries. Mobile databases differ significantly from server-class storage in terms of platform, usage, and measurement. Phones are multi-tenant, end-user devices that the database must share with other apps. Contrary to traditional database design goals, workloads on phones are single-app, bursty, and rarely saturate the CPU. We argue that mobile storage design should refocus on what matters on the mobile platform: latency and energy. As accurate performance measurement tools are necessary to evaluation of good database design, this uncovers another issue: Traditional database benchmarking methods produce misleading results when applied to mobile devices, due to evaluating performance at saturation. Development of databases and measurements specifically designed for the mobile platform is necessary to optimize user experience of the most common database usage in the world.

Introduction.

There are 2 billion+ smartphones on Earth, each averaging 2 queries per second [2]. Databases furnish an abstraction to query persistent storage in a structured fashion. Data management is a bottleneck for mobile workloads [5], so improving mobile databases performance is critical. To do so, we need to understand the environment and behavior of smartphone databases. There are several key differences between embedded platforms and traditional database environments that affect DB behavior and measurement. We outline these differences, and show how traditional database designs and measurement metrics focus on the wrong things.

The pH is Too High: Database guarantees need to become less Acid and more Basic.

Phone databases default to correct over performant behavior, for example by implementing full ACID guarantees. The costs can be significant: SQLite, the installed database on Android, by default implements file locking enabled to preserve

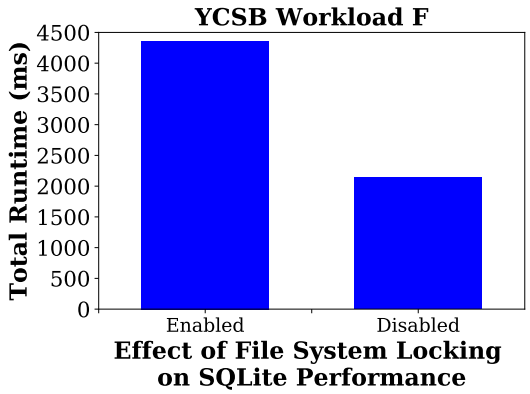

Figure 1: The cost of making SQLite thread safe

isolation[3]. Figure 1 shows the performance gain when running YCSB workload F [1] that can be had by eliminating this single safeguard.

Smartphones can harvest the performance gains from implementing a simpler set of safety guarantees. To be sure, phone databases will still need to assure data integrity (Atomicity) and storage invariants (Consistency).

But do we really need Durability on current phones? Batteries are typically soldered in, so the chance of an abrupt power cut is remote. The Isolation guarantee is unnecessary in most cases. While modern server-class databases, to maximize throughput, support simultaneous connections, smartphone databases are per-app. Additionally, apps have low throughput requirements. We speculate most phone apps never implement internal database threading, and that the threading and locking overhead can be eliminated from most phone databases.

Workloads are bursty.

Server-class databases must expect continuous operation requests; transaction rate is a key performance metric[4]. Embedded databases, however, should not: Figure 2 shows the irregular bursts of a representative query pattern. Indeed, as we discuss later, focusing on throughput during benchmarking of database performance can produce misleading results. A tpm of 112k can be obtained with dedicated hardware costing $100k+ [4]. Yet, Figure 2 shows rate bursts of 36k tpm on phones costing $500 – 1/3 the rate for less than .1% the cost. The key is that phones do not need to *sustain* this rate.

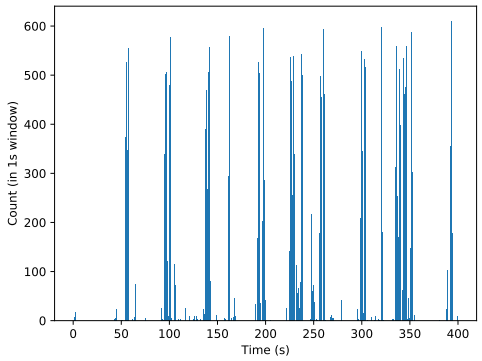

Figure 2: Queries over time

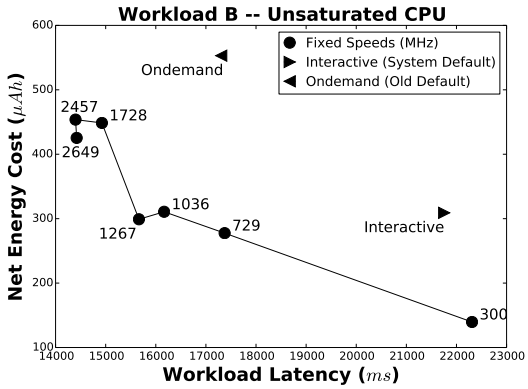

Figure 3: Current default suboptimal energy and latency performance

Rather, latency is paramount: mobile is an end-user platform, and latency directly affects app responsiveness. Developers of mobile databases, given the limited platform, cannot rely on server-grade hardware for a solution. They need to focus on designing DB engines to optimize burst response, with smart cache pre-fetching to yield low latency for the duration of the current burst, and a large enough caches to serve the next burst request. They can safely discount access patterns exhausting the cache – the activity burst will likely end before that.

Mobile power and performance are suboptimal.

Resource-limited phones are particularly concerned with power usage, and power-performance tradeoffs are a common study. Surprisingly, the default settings on Android are often not optimal for database usage. Figure 3 shows the power-performance results for YCSB workload B for a number of different CPU policy settings. Several non-default settings outperform the system defaults (Ondemand and Interactive) in *both* areas. We have found that the optimal CPU

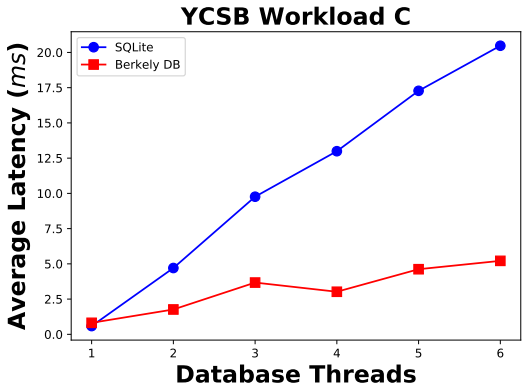

Figure 4: SQLite-BDB Performance Comparison

policy depends strongly on the nature of the database workload – in most cases studied, the setting is a non-default one. There is thus significant room for improvement in this area. Likely, implementation of better system policies, advised by database workload semantics, will yield increased database performance and/or decreased energy cost.

Mobile needs its own benchmarks.

In order to evaluate and tune any database, representative benchmarks are necessary. The different conditions and aims of mobile platforms, however, mean that traditional benchmark metrics can produce misleading results. Figure 4 illustrates a basic performance comparison of how two different databases scale under increasing numbers of workloads (YCSB-C). BDB, which scales better, would normally be seen as the better alternative. However, for mobile, databases are per-app, and it is the base single-thread case that is most relevant. SQLite performs the better in this study, and indeed in most of the YCSB workloads and conditions that we have studied on phones. The standard benchmark evaluation would produce misleading results.

REFERENCES

[1] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *SOCC*.

[2] Oliver Kennedy, Jerry Antony Ajay, Geoffrey Challen, and Lukasz Ziarek. 2015. Pocket Data: The Need for TPC-MOBILE. In *TPC-TC*.

[3] sqlite.org. 2019. Sqlite. (2019). https://www.sqlite.org/tempfiles.html

[4] tpc.org. 2018. TPC. (2018). http://www.tpc.org/tpcc/results/tpcc_results.asp

[5] Yan Wang and Atanas Rountev. 2016. Profiling the responsiveness of Android applications via automated resource amplification. In *MOBILESoft*. ACM, 48–58.