Enhancement of the Performance of Small Footprint Database and Open Source Monitoring Tool

Sukh Deo

Under the guidance of

Prof. Deepak B. Phatak
Dept. of CSE
IIT Bombay

October 17, 2013
Table of contents

- Introduction
  - Motivation
  - Introduction of Small Footprint Databases
  - Introduction of Monitoring Tool
- Literature Survey on Small Footprint Databases
  - Berkeley DB
  - Perst DB
  - SQLite DB
  - Comparison among Berkeley DB, Perst DB, and SQLite DB
  - Journal Mode in SQLite and Shadow Paging
- Literature Survey on Monitoring Tool
  - IOzone
  - AndroBench
  - MobiBench
- Problem Statement and Proposals
- Conclusion and Future Work
Motivation

- Todays mobile devices are multifunctional devices
- Mobile devices are siblings of personal computers
- People want to use many applications on mobile device
- Mobile device has some limitation
  - Small memory
  - Battery issues
- Enhance the performance of small footprint databases and monitoring tool
Motivation

- Todays mobile devices are multifunctional devices
- Mobile devices are siblings of personal computers
- People want to use many applications on mobile device
- Mobile device has some limitation
  - Small memory
  - Battery issues
- Enhance the performance of small footprint databases and monitoring tool
Motivation

- Today's mobile devices are multifunctional devices.
- Mobile devices are siblings of personal computers.
- People want to use many applications on mobile device.
- Mobile device has some limitation:
  - Small memory
  - Battery issues
- Enhance the performance of small footprint databases and monitoring tool.
Motivation

- Today's mobile devices are multifunctional devices
- Mobile devices are siblings of personal computers
- People want to use many applications on mobile device
- Mobile device has some limitation
  - Small memory
  - Battery issues
- Enhance the performance of small footprint databases and monitoring tool
Motivation

- Today's mobile devices are multifunctional devices
- Mobile devices are siblings of personal computers
- People want to use many applications on mobile device
- Mobile device has some limitation
  - Small memory
  - Battery issues
- Enhance the performance of small footprint databases and monitoring tool
Small footprint databases

- Called lightweight databases or embedded databases
- Specially designed for mobile devices
- People want to use many applications on mobile devices
- Mobile devices require such types of database that uses small memory and low power
- Example, Berkeley DB, Perst DB, SQLite, and etc.
Small footprint databases

- Called lightweight databases or embedded databases
- Specially designed for mobile devices
- People want to use many applications on mobile device
- Mobile devices require such types of database that uses small memory and low power
- Example, Berkeley DB, Perst DB, SQLite, and etc.
Small footprint databases

- Called lightweight databases or embedded databases
- Specially designed for mobile devices
- People want to use many applications on mobile device
- Mobile devices require such types of database that uses small memory and low power
  - Example, Berkeley DB, Perst DB, SQLite, and etc.
Small footprint databases

- Called lightweight databases or embedded databases
- Specially designed for mobile devices
- People want to use many applications on mobile device
- Mobile devices require such types of database that uses small memory and low power
- Example, Berkeley DB, Perst DB, SQLite, and etc.
Small footprint databases

- Called lightweight databases or embedded databases
- Specially designed for mobile devices
- People want to use many applications on mobile devices
- Mobile devices require such types of database that uses small memory and low power
- Example, Berkeley DB, Perst DB, SQLite, and etc.
In mobile device, each application generates the specific load
Measures the load of each application that are running on mobile device
For finding the correct load of application, we have to analyze data types and file system of executing application.
For measure the performance of mobile device, we need mobile benchmarking tool
Example, IOzone, AndroBench, and Mobibench
Monitoring Tool

- In mobile device, each application generates the specific load
- Measures the load of each application that are running on mobile device
- For finding the correct load of application, we have to analyze data types and file system of executing application.
- For measure the performance of mobile device, we need mobile benchmarking tool
- Example, IOzone, AndroBench, and Mobibench
In mobile device, each application generates the specific load

Measures the load of each application that are running on mobile device

For finding the correct load of application, we have to analyze data types and file system of executing application.

For measure the performance of mobile device, we need mobile benchmarking tool

Example, IOzone, AndroBench, and Mobibench
In mobile device, each application generates the specific load.

Measures the load of each application that are running on mobile device.

For finding the correct load of application, we have to analyze data types and file system of executing application.

For measure the performance of mobile device, we need mobile benchmarking tool.

Example, IOzone, AndroBench, and Mobibench.
In mobile device, each application generates the specific load.

Measures the load of each application that are running on mobile device.

For finding the correct load of application, we have to analyze data types and file system of executing application.

For measure the performance of mobile device, we need mobile benchmarking tool.

Example, IOzone, AndroBench, and Mobibench.
Literature Survey on Small Footprint Databases
- Berkeley DB is an embedded database
- Open source and written in C language
- Used to store the data, key/value technique
- Example, to store the user name and password
Berkeley Database

- Berkeley DB is an embedded database
- Open source and written in C language
- Used to store the data, key/value technique
- Example, to store the user name and password
Berkeley Database

- Berkeley DB is an embedded database
- Open source and written in C language
- Used to store the data, key/value technique
- Example, to store the user name and password
Berkeley DB is an embedded database
- Open source and written in C language
- Used to store the data, key/value technique
- Example, to store the user name and password
Access method in Berkeley DB

- B+tree
- Hash
- Fixed or variable length record
Access method in Berkeley DB

- B+tree
- Hash
- Fixed or variable length record
Access method in Berkeley DB

- B+tree
- Hash
- Fixed or variable length record
Limitations

- It does not support RDBMS
- It has no schema
- It stores only the record key/value technique, so that we cannot compare the data other than key value
Limitations

- It does not support RDBMS
- It has no schema
- It stores only the record key/value technique, so that we cannot compare the data other than key value
Limitations

- It does not support RDBMS
- It has no schema
- It stores only the record key/value technique, so that we cannot compare the data other than key value
Perst Database is an open source embedded databases, based on Java object oriented architecture.

Feature are:
- B.tree
- R-tree index
- Specialized versions of collections for thick indices and bit indices
Perst DB

- Perst Database is an open source embedded databases, based on Java object oriented architecture
- Feature are:
  - B.tree
  - R-tree index
  - Specialized versions of collections for thick indices and bit indices
Perst Database is an open source embedded databases, based on Java object oriented architecture.

Feature are:
- B.tree
- R-tree index
- Specialized versions of collections for thick indices and bit indices
Perst Database is an open source embedded databases, based on Java object oriented architecture.

Feature are:
- B.tree
- R-tree index
- Specialized versions of collections for thick indices and bit indices
Limitation

- It is application specific database for storing real object information.
- Example, we can use Perst DB for restaurant searching application, because we can access and store information using making object.
Limitation

- It is application specific database for storing real object information.
- Example, we can use Perst DB for restaurant searching application, because we can access and store information using making object.
Due to limitation of small memory, it does not provide all the features of RDBMS.

Some other features that are not present in RDBMS are zero configuration, self-content, and server-less database.

It has a schema, thus we can design our own tables as we want.
• Due to limitation of small memory, it does not provide all the features of RDBMS

• Some other features that are not present in RDBMS are zero configuration, self-content, and server-less database

• It has a schema, thus we can design our own tables as we want
SQLite DB

- Due to limitation of small memory, it does not provide all the features of RDBMS.
- Some other features that are not present in RDBMS are zero configuration, self-content, and server-less database.
- It has a schema, thus we can design our own tables as we want.
Architecture of SQLite

Figure: Architecture of SQLite
SQLite is a native database. It has a schema, so that we can design our own tables as we want

Berkeley DB is an embedded database. It stores the data by key/value schema

Perst DB is an object oriented, application specific database for storing real object information

We cannot really compare properly, because each has some special property
Comparison among Berkeley DB, Perst DB, and SQLite

- SQLite is a native database. It has a schema, so that we can design our own tables as we want.
- Berkeley DB is an embedded database. It stores the data by key/value schema.
- Perst DB is an object oriented, application specific database for storing real object information.
- We cannot really compare properly, because each has some special property.
Comparison among Berkeley DB, Perst DB, and SQLite DB

- SQLite is a native database. It has a schema, so that we can design our own tables as we want.
- Berkeley DB is an embedded database. It stores the data by key/value schema.
- Perst DB is an object oriented, application specific database for storing real object information.
- We cannot really compare properly, because each has some special property.
Comparison among Berkeley DB, Perst DB, and SQLite DB

- SQLite is a native database. It has a schema, so that we can design our own tables as we want.
- Berkeley DB is an embedded database. It stores the data by key/value schema.
- Perst DB is an object oriented, application specific database for storing real object information.
- We cannot really compare properly, because each has some special property.
Journal Mode of SQLite

- Rollback journal
- Write-Ahead Log (WAL) Mode
- OFF Mode
Rollback journal

Write-Ahead Log (WAL) Mode

OFF Mode
Journal Mode of SQLite

- Rollback journal
- Write-Ahead Log (WAL) Mode
- OFF Mode
Figure: commit process
Figure: commit process
Figure: commit process
**Figure:** commit process
Figure: commit process
Figure: commit process
Figure: commit process
Figure: commit process
Figure: commit process
Figure: commit process
Write-Ahead Log (WAL) Mode

- The WAL approach inverts of Rollback journal
- The original content is preserved in the database file and the changes are appended into a separate WAL file
- COMMIT can happen without ever writing to the original database
- Allows readers to continue operating from the original unaltered database while changes are simultaneously being committed into the WAL
Write-Ahead Log (WAL) Mode

- The WAL approach inverts of Rollback journal
- The original content is preserved in the database file and the changes are appended into a separate WAL file
- COMMIT can happen without ever writing to the original database
- Allows readers to continue operating from the original unaltered database while changes are simultaneously being committed into the WAL
Write-Ahead Log (WAL) Mode

- The WAL approach inverts of Rollback journal
- The original content is preserved in the database file and the changes are appended into a separate WAL file
- COMMIT can happen without ever writing to the original database
- Allows readers to continue operating from the original unaltered database while changes are simultaneously being committed into the WAL
Write-Ahead Log (WAL) Mode

- The WAL approach inverts Rollback journal
- The original content is preserved in the database file and the changes are appended into a separate WAL file
- COMMIT can happen without ever writing to the original database
- Allows readers to continue operating from the original unaltered database while changes are simultaneously being committed into the WAL
Write-Ahead Log (WAL) Mode

**Advantage**
- WAL provides more concurrency as readers do not block writers and a writer does not block readers.
- WAL uses fewer fsync() operations and is thus less vulnerable to problems on systems where the fsync() system call is broken.

**Disadvantage**
- All processes using a database must be on the same host computer. WAL does not work over a network file system.
- WAL might be very slightly slower than the traditional rollback-journal approach in applications that do mostly reads and seldom write.
- There is the extra operation of checkpointing.
Write-Ahead Log (WAL) Mode

- **Advantage**
  - WAL provides more concurrency as readers do not block writers and a writer does not block readers
  - WAL uses fewer fsync() operations and is thus less vulnerable to problems on systems where the fsync() system call is broken

- **Disadvantage**
  - All processes using a database must be on the same host computer. WAL does not work over a network file system
  - WAL might be very slightly slower than the traditional rollback-journal approach in applications that do mostly reads and seldom write
  - There is the extra operation of check pointing
OFF Mode

- OFF mode journal used when no need of page replacement technique
- Example, while playing the game no need of such page replacement technique, because that time only speed matter. So that, we can use OFF mode journal
OFF Mode

- OFF mode journal used when no need of page replacement technique
- Example, while playing the game no need of such page replacement technique, because that time only speed matter. So that, we can use OFF mode journal
Shadow paging

- Database maintains two images per page during the transaction, i.e., shadow page and a new page.
- The shadow page remains the same throughout the duration of the transaction.
- The new page will be changed when a transaction performs a write operation.
- To commit modification, it modifies all pointers to old (shadow) page to now point to new page, and frees the shadow page.
Shadow paging

- database maintains two images per page during the transaction, i.e., shadow page and a new page
- The shadow page is remain same the duration of the transaction
- The new page will be changed when a transaction performs a write operation
- To commit modification, it modifies all pointers to old (shadow) page to now point to new page, and frees the shadow page
Shadow paging

- Database maintains two images per page during the transaction, i.e., shadow page and a new page.
- The shadow page is remain same the duration of the transaction.
- The new page will be changed when a transaction performs a write operation.
- To commit modification, it modifies all pointers to old (shadow) page to now point to new page, and frees the shadow page.
Shadow paging

- database maintains two images per page during the transaction, i.e., shadow page and a new page
- The shadow page is remain same the duration of the transaction
- The new page will be changed when a transaction performs a write operation
- To commit modification, it modifies all pointers to old (shadow) page to now point to new page, and frees the shadow page
Literature Survey on Monitoring Tool
Literature Survey on Monitoring Tool

• IOzone
• AndroBench
• MobiBench
IOzone
AndroBench
MobiBench
Literature Survey on Monitoring Tool

- IOzone
- AndroBench
- MobiBench
IO Characteristics

- IO type (Read vs. Write)
- IO size (KB)
- Spatial locality (Sequential vs. Random)
- Temporal locality (Hot vs. Cold)
IO Characteristics

- IO type (Read vs. Write)
- IO size (KB)
- Spatial locality (Sequential vs. Random)
- Temporal locality (Hot vs. Cold)
IO Characteristics

- IO type (Read vs. Write)
- IO size (KB)
- Spatial locality (Sequential vs. Random)
- Temporal locality (Hot vs. Cold)
IO Characteristics

- IO type (Read vs. Write)
- IO size (KB)
- Spatial locality (Sequential vs. Random)
- Temporal locality (Hot vs. Cold)
IOzone

- IOzone is benchmark tool for file system
- It is not open source tool
- Used by computer vendors to measures the IO performance of accessing read, write, fread, and fwrite operations of a single file
- IOzone is benchmark tool for file system
- It is not open source tool
- Used by computer vendors to measure the IO performance of accessing read, write, fread, and fwrite operations of a single file
Limitations

- It does not include fsync(). So, that we cannot measure the performance of SQLite.
- In Android devices, there is no way of getting the superuser permission in user level for remounting and unmounting file, but IOzone need to superuser permission. So, it cannot measure the performance of Android devices.
Limitations

- It does not include fsync(). So, that we cannot measure the performance of SQLite.
- In Android devices, there is no way of getting the superuser permission in user level for remounting and unmounting file, but IOzone need to superuser permission. So, it cannot measure the performance of Android devices.
AndroBench

AndroBench is a storage benchmarking tool for Android mobile devices

- It is not open source tool
- It measures the sequential and random IO performance and throughput of various SQLite transaction, i.e., insert, delete, and update
AndroBench is a storage benchmarking tool for Android mobile devices.

- It is not an open source tool.
- It measures the sequential and random IO performance and throughput of various SQLite transactions, i.e., insert, delete, and update.
Structure of AndroBench

Target Partition
Select testing partition

File Size
Change read/write file size

Buffer Size
Change buffer size for sequential & random access

Number of Transactions
Change number of transaction for SQLite benchmark

Setting tab
AndroBench
Storage Benchmarking Tool

Set partition:
/data/

File size:
RD:32 WR:2 MB

Buffer size:
SEQ:256 RND:4 KB

Transactions:
300
Limitations

- It has no method to measure the performance of `fsync()` call
- It does not allow changing synchronization option such as `O_SYNC`, `O_DIRECT`, and `Mmap`
Limitations

- It has no method to measure the performance of fsync() call
- It does not allow changing synchronization option such as O SYNC, O DIRECT, and Mmap
MobiBench

- Mobibench is a storage performance benchmarking tool
- It is open source tool
- It combines both characteristics of IOzone and AndroBench
- It is implemented in two versions, first one shell application and second one is Android application
- It is implemented in C
MobiBench

- Mobibench is a storage performance benchmarking tool
- It is an open source tool
- It combines both characteristics of IOzone and AndroBench
- It is implemented in two versions, first one is shell application and second one is Android application
- It is implemented in C
MobiBench

- Mobibench is a storage performance benchmarking tool
- It is open source tool
- It combines both characteristics of IOzone and AndroBench
- It is implemented in two versions, first one shell application and second one is Android application
- It is implemented in C
MobiBench

- Mobibench is a storage performance benchmarking tool
- It is open source tool
- It combines both characteristics of IOzone and AndroBench
- It is implemented in two versions, first one shell application and second one is Android application
- It is implemented in C
- Mobibench is a storage performance benchmarking tool
- It is open source tool
- It combines both characteristics of IOzone and AndroBench
- It is implemented in two versions, first one shell application and second one is Android application
- It is implemented in C
Measure Tab of MobiBench
Setting Tab of MobiBench

![MobiBench Setting Tab](image)

- **Partition**: /data
- **Thread number**: 1 (1-100)
- **File I/O**
  - Seq. Write
  - Seq. Read
  - Ran. Write
  - Ran. Read
- **SQLite**
  - Insert
  - Update
Problem Statement and Proposal
Problem on SQLite

- Flash memory has two critical drawbacks
  - A segment, blocks of flash memory, need to be erased before they can be rewritten
  - The life of each memory block is limited to 10,00,000 writes
- SQLite uses rollback journal or write ahead logging, which are update in place approach
- Due to erase first than update, it decrease the performance
- Rollback journal does not provides to overcome the above two mention problem of flash memory
Problem on SQLite

- Flash memory has two critical drawbacks
  - A segment, blocks of flash memory, need to be erased before they can be rewritten
  - The life of each memory block is limited to 10,00,000 writes
- SQLite uses rollback journal or write ahead logging, which are update in place approach
- Due to erase first than update, it decreases the performance
- Rollback journal does not provide to overcome the above two mention problem of flash memory
Problem on SQLite

- flash memory has two critical drawbacks
  - A segment, blocks of flash memory, need to be erased before they can be rewritten
  - The life of each memory block is limited to 10,00,000 writes
- SQLite uses rollback journal or write ahead logging, which are update in place approach
- Due to erase first than update, it decrease the performance
- Rollback journal does not provides to overcome the above two mention problem of flash memory
Problem on SQLite

- flash memory has two critical drawbacks
  - A segment, blocks of flash memory, need to be erased before they can be rewritten
  - The life of each memory block is limited to 10,00,000 writes
- SQLite uses rollback journal or write ahead logging, which are update in place approach
- Due to erase first than update, it decreases the performance
- Rollback journal does not provide to overcome the above two mention problem of flash memory
Problem on SQLite

- flash memory has two critical drawbacks
  - A segment, blocks of flash memory, need to be erased before they can be rewritten
  - The life of each memory block is limited to 10,00,000 writes
- SQLite uses rollback journal or write ahead logging, which are update in place approach
- Due to erase first than update, it decrease the performance
- Rollback journal does not provides to overcome the above two mention problem of flash memory
Shadow paging is better solution

No need to erase immediately, we simply update the data in free space

After commit, it modifies all pointers to old (shadow) page to now point to new page and garbage collector will collect the old page
Shadow paging is better solution
No need to erase immediately, we simply update the data in free space
After commit, it modifies all pointers to old (shadow) page to now point to new page and garbage collector will collect the old page
• Shadow paging is better solution
• No need to erase immediately, we simply update the data in free space
• After commit, it modifies all pointers to old (shadow) page to now point to new page and garbage collector will collect the old page
Possible optimization on MobiBench

Problem on MobiBench
- MobiBench has no procedure to analyze the IO characteristics of Hot and Cold chunk
- It has no procedures to measure shadow paging technique

Proposal
- Since, MobiBench is open source benchmarking tool, we can add a module that will measure the performance of Shadow paging
Possible optimization on MobiBench

- **Problem on MobiBench**
  - MobiBench has no procedure to analyze the IO characteristics of Hot and Cold chunk
  - It has no procedures to measure shadow paging technique

- **Proposal**
  - Since, MobiBench is open source benchmarking tool, we can add a module that will measure the performance of Shadow paging
Problem on MobiBench
- MobiBench has no procedure to analyze the IO characteristics of Hot and Cold chunk
- It has no procedures to measure shadow paging technique

Proposal
- Since, MobiBench is open source benchmarking tool, we can add a module that will measure the performance of Shadow paging
Problem on MobiBench

- MobiBench has no procedure to analyze the IO characteristics of Hot and Cold chunk
- It has no procedures to measure shadow paging technique

Proposal

- Since, MobiBench is open source benchmarking tool, we can add a module that will measure the performance of Shadow paging
Conclusion

- We have discussed Small footprint database, i.e., Berkeley DB, Perst DB, and SQLite
- We found that SQLite gives better performance
- We have discussed architecture and journal mode of SQLite, i.e., rollback, WAL, and OFF mode
- We have also discussed possible way to optimize the performance of SQLite, i.e., Shadow paging
- We have discussed different monitoring tool, i.e., IOzone, AndroBench, and MobiBench
- We found that Mobibench give better performance
- We proposed to add a modules in MobiBench for measure the performance of Shadow paging
Conclusion

- We have discussed Small footprint database, i.e., Berkeley DB, Perst DB, and SQLite
- We found that SQLite gives better performance
- We have discussed architecture and journal mode of SQLite, i.e., rollback, WAL, and OFF mode
- We have also discussed possible way to optimize the performance of SQLite, i.e., Shadow paging
- We have discussed different monitoring tool, i.e., IOzone, AndroBench, and MobiBench
- We found that MobiBench gives better performance
- We proposed to add a module in MobiBench for measure the performance of Shadow paging
Conclusion

- We have discussed Small footprint database, i.e., Berkeley DB, Perst DB, and SQLite
- We found that SQLite gives better performance
- We have discussed architecture and journal mode of SQLite, i.e., rollback, WAL, and OFF mode
- We have also discussed possible way to optimize the performance of SQLite, i.e., Shadow paging
- We have discussed different monitoring tool, i.e., IOzone, AndroBench, and MobiBench
- We found that Mobibench give better performance
- We proposed to add a module in MobiBench for measure the performance of Shadow paging
We have discussed Small footprint database, i.e., Berkeley DB, Perst DB, and SQLite.

We found that SQLite gives better performance.

We have discussed architecture and journal mode of SQLite, i.e., rollback, WAL, and OFF mode.

We have also discussed possible way to optimize the performance of SQLite, i.e., Shadow paging.

We have discussed different monitoring tool, i.e., IOzone, AndroBench, and MobiBench.

We found that Mobibench give better performance.

We proposed to add a modules in MobiBench for measure the performance of Shadow paging.
Conclusion

- We have discussed Small footprint database, i.e., Berkeley DB, Perst DB, and SQLite.
- We found that SQLite gives better performance.
- We have discussed architecture and journal mode of SQLite, i.e., rollback, WAL, and OFF mode.
- We have also discussed possible way to optimize the performance of SQLite, i.e., Shadow paging.
- We have discussed different monitoring tool, i.e., IOzone, AndroBench, and MobiBench.
- We found that Mobibench give better performance.
- We proposed to add a modules in MobiBench for measure the performance of Shadow paging.
Conclusion

- We have discussed Small footprint database, i.e., Berkeley DB, Perst DB, and SQLite
- We found that SQLite gives better performance
- We have discussed architecture and journal mode of SQLite, i.e., rollback, WAL, and OFF mode
- We have also discussed possible way to optimize the performance of SQLite, i.e., Shadow paging
- We have discussed different monitoring tool, i.e., IOzone, AndroBench, and MobiBench
- We found that Mobibench give better performance
- We proposed to add a modules in MobiBench for measure the performance of Shadow paging
Conclusion

- We have discussed Small footprint database, i.e., Berkeley DB, Perst DB, and SQLite.
- We found that SQLite gives better performance.
- We have discussed architecture and journal mode of SQLite, i.e., rollback, WAL, and OFF mode.
- We have also discussed possible ways to optimize the performance of SQLite, i.e., Shadow paging.
- We have discussed different monitoring tools, i.e., IOzone, AndroBench, and MobiBench.
- We found that MobiBench gives better performance.
- We proposed to add a module in MobiBench for measuring the performance of Shadow paging.
Future work

- Implement shadow paging in SQLite
- Change the source code of MobiBench, so that we can compare the performance between shadow paging and WAL
Future work

- Implement shadow paging in SQLite
- Change the source code of MobiBench, so that we can compare the performance between shadow paging and WAL
Thank You
References

- https://www.sqlite.org/
- http://dl.acm.org/
- http://www.mcobject.com/perst