**Code Optimization in VanillaDb Core**

Version 1.0

February 2014

TY Feng, YS Lin

NetDB

SQL

1. **Default instance of different types of constant**

Class:

org.vanilladb.core.sql.Constant

Hotspot:

defaultInstance(Type):Constant

Optimization:

For the method defaultInstance(Type), we materialized its output values. For each type of the constant, there is an instance with default value stored as static member in the Constant.

Code committed date:

2013-08-23

**2. The fields set of schema**

Class:

org.vanilladb.core.sql.Schema

Hotspot:

fields():SortedSet<String>

Optimization:

Creating a tree set of fields in a schema is costly. We materialized its output when this schema instance is created. However, the schema instance is mutable. We need to update the materialized field set when the schema changes.

Code committed date:

2013-08-23

**3. The number of bytes per char**

Class:

org.vanilladb.core.sql.VarcharType

Hotspot:

maxSize():int

Optimization:

Computing the maximal number of bytes to storing a given type of varchar constant requires getting the encoding information of the specified char set. We stored the value of number of bytes as a static member.

Code committed date:

2013-08-23

Index

1. **Search algorithm in the directory block of B-Tree**

Class:

org.vanilladb.core.storage.index.btree.BTreeDir

Hotspot:

findSlotBefore(Constant):int

Optimization:

The original algorithm for searching a slot in the directory block is sequential search. The time complexity of the sequential search is O(N) and N stands for the number of slots in the target block.

We use the binary search to speed up the search process. The time complexity of binary search is O(logN).

Code committed date:

2014-02-28

1. **Search algorithm in the leaf block of B-Tree**

Class:

org.vanilladb.core.storage.index.btree.BTreeLeaf

Hotspot:

moveSlotBefore():void

Optimization:

BTreeLeaf use the same algorithm in moveSlotBefore as BTreeDir. We also use the binary search to speed up the search process. The time complexity of binary search is O(logN).

Code committed date:

2014-02-28

1. **The number of records of B-Tree page**

Class:

org.vanilladb.core.storage.index.btree.BTreePage

Hotspot:

getNumRecords():int

Optimization:

Materialize the number of records of B-Tree Page.

Code committed date:

2014-02-28

Metadata

1. **The table information**

Class:

org.vanilladb.core. storage.metadata.TableMgr

Hotspot:

getTableInfo():TableInfo

Optimization:

Materialize the output table information.

Code committed date:

2013-08-27

1. **The index information**

Class:

org.vanilladb.core. storage.metadata.IndexMgr

Hotspot:

getIndexInfo(): Map<String, IndexInfo>

Optimization:

Materialize the output index information.

Code committed date:

2013-11-18

1. **Split MetadataMgr into CatalogMgr and StatMgr**

Class:

org.vanilladb.core. storage.metadata.MetaDataMgr

->org.vanilladb.core. storage.metadata.CatalogMgr

org.vanilladb.core. storage.metadata.statistics.StatMgr

Optimization:

The logical recovery process needs to access table info and index info, so the catalog manager should be initialized before system recovery.

The statistic manager which builds the histogram should be initialized after system recovery.

We split two functions of original metadata manager to catalog manager and statistics manager.

Code committed date:

2013-11-21

File

1. **Default instances of different types of constant**

Class:

org.vanilladb.core. storage.file.BlockId

Hotspot:

toString():String

hashCode():int

Optimization:

Materialize the output string of toString and its hashed value.

Code committed date:

2013-08-23

**2. The size of data file**

Class:

org.vanilladb.core. storage.file.FileMgr

Hotspot:

size(String):long

Optimization:

The method size will get the current size of the specified file channel. The invocation of java.nio.channels.FileChannel.size is cost. We materialize the size information of each data file. When the new block has been appended to a given file, we need to update the size information accordingly.

Code committed date:

2013-08-23

Log

1. **Pointer size in log record**

Class:

org.vanilladb.core. storage.log.LogMgr

org.vanilladb.core. storage.log.LogIterator

Hotspot:

Page.maxSize(IntegerConstant) invoked by computing the size of pointer in each log record.

Optimization:

Materializes the number of bytes used to store a pointer in log record.

Code committed date:

2013-08-23

Record

1. **EMPTY and INUSE flags in each record slot**

Class:

org.vanilladb.core. storage.record.RecordPage

Hotspot:

IntegerConstant(INUSE), IntegerConstant(EMPTY)

Optimization:

Materialize the constant value of slot flags.

Code committed date:

2013-08-23

1. **The size of EMPTY and INUSE flags**

Class:

org.vanilladb.core. storage.record.RecordPage

Hotspot:

Page.maxSize(INTEGER)

Optimization:

Materialize the number of bytes to store a slot flag.

Code committed date:

2013-08-23

1. **The offset of each field in a record**

Class:

org.vanilladb.core. storage.record.RecordPage

Hotspot:

Page.maxSize(sch.type(fldname))

Optimization:

Materialize the offset map in RecordPage’s constructor.

Code committed date:

2013-08-23

Tx.Concurrency

1. **Release all locks**

Class:

org.vanilladb.core. storage.tx.concurrency.LockTable

Hotspot:

releaseAll(long, Boolean): void

Optimization:

Add a data structure to keep track of the mapping of locked objects and transactions.

Code committed date:

2014-2-24

Tx.Recovery

1. **Logical recovery on index structure**

Class:

org.vanilladb.core.storage.tx.recovery.RecoveryMgr, IndexDeleteRecord, IndexInsertRecord, IndexRecoveryUtil

Index

Hotspot:

The original logging record for index structure composites block-level information (e.g., [blk 1, offset 20, oldval 5, newval 7]. This makes the modification on index structure writing a lot of log records.

Optimization:

Use logical log record instead of byte information log record. Define new types of log record. Change the recovery process to 3-phases, analyze, redo and undo. (Reference: CH18-6 of Database Management System 3/e, Ramakrishnan.)

Note:

The logical recovery process needs to access table info and index info, so the catalog manager should be initialized before system recovery.

The statistic manager which builds the histogram should be initialized after system recovery.

Code committed date:

2013-11-21

Transaction

1. **Increase the concurrency level of TransactionMgr**

Class:

org.vanilladb.core.storage.tx.TransactionMgr

Hotspot:

All public method of TransactionMgr are surrounded by the keyword “Synchronized”. It causes that only one thread can execute in TransactionMgr, even if other thread want to do the totally irrelevant things.

Optimization:

We reduce the size of region surrounded by “Synchronized” and use the different latch (by synchronized on other object) for different actions.

Code committed date:

2014-2-21