Transaction

www.iqlect.com

Copyright © 2014 IQLECT All rights reserved

1. Overview

Transaction ensures that set of db operations, such as reads or writes, are handled by the db as a unit of work. Which means that the unit of work is either successful or failure when user is done with the operations. This ensures that user gets the right view of the db snapshot and alters the view in consistent manner, let other users deal with their unit of work simultaneously and finally make the changes permanent to the db. This property is called ACID where A stands for atomicity, C for consistency, I for isolation and D for durable. The transaction allows users to run concurrently on a db while ensuring the ACID property at the same time. Hence transaction improves the performance of the db as well

BangDB provides ACID property to the user through transaction. BangDB offers the transaction as configuration parameter and user can run on the same db in transactional or non-transactional mode based on the setting

For mature key value store, transaction is a necessary feature and is a must for many use case scenarios apart from the usual known such as transactional website, financial transaction, e-commerce etc. Hence having a robust and high performance transaction support in the db is critical as it supports wide range of use case scenarios and at the same time act as major differentiating factor

2. Transaction in BangDB

At the core, BangDB is designed for all single operations to ensure the ACID property. Which means that multiple user connections can be created and many simultaneous operations (reads and writes) can be run with worrying about the concurrency issues. This is because BangDB is a concurrent database engine which allows multiple threads or connections to modify the db at a time. This is possible due to the design of the db (please see the architecture doc for more info) which takes locks at the appropriate places to ensure sanity. It is interesting to note that even in highest possible concurrent situation, whole db is never locked. In fact whole index is also never locked. On an average two pages (among thousands of pages) are locked for write operation and in read we avoid the locking fully in most of the cases. This enhances the performance of db to great extent as it can leverage the cores of the machines in optimal manner

However, for multiple operations to be treated as a single unit of work, we need more than just the single op ACID guarantee from the db. And for this reason we explicitly need a transactional boundary within which we can place multiple operations and play with them yet keeping the ACID property intact. BangDB provides the support by implementing optimistic concurrency control (occ). Note that for single operation, the concurrency is done by locking and for multiple operations as a unit of work, the concurrency is offered through transaction (occ)

The transaction can be enabled or disabled by setting the appropriate configuration property in the bangdb.config file or opening the db with appropriate flag (DB_OPTIMISTIC_TRANSACTION). There is another setting in the config file known as AUTOCOMMIT, which when enabled allows user to run single op transaction in an implicit manner and if disabled, requires user to use explicit transaction boundary

3. Implementation Details

As stated above the transaction support is provided in BangDB using occ (optimistic concurrency control). There are mainly two ways of achieving the transaction property, one is through locking and other is though the occ. While locking way to provide the ACID support is also called the pessimistic way of achieving the same. This is because no matter what db would lock the resources as soon as it's fetched from the db whereas in occ locking is required only for small period and that too when actual update is being performed. This

Copyright © 2014 IQLECT All rights reserved

enables the db to allow multiple concurrent transaction to proceed and yet ensuring the full ACID. The optimistic way of doing it assumes that most of the time other concurrent transaction will not intersect with a particular ongoing transaction, which is true in general use cases. For example, what would be the probability of multiple transactions accessing same page from available 100 thousand pages? But even if it happens that two or more transactions intersect, then the one or more transactions are aborted and then user will have to retry to updates. The retry is the only penalty user pays in the occ approach. But this is not too often and over the period of time overall of gains from the occ outshine this retry downside of the approach

- ***Optimistic Concurrency Control (OCC)***

  OCC is a method to control concurrency for db that assumes that multiple transactions can complete without affecting each other and therefore transaction can proceed without locking the data resources that they affect or change. Before committing the changes to the db, each transaction would run verifications on their data and if the result of verification is positive then the transaction is rolled back or aborted. This is mostly suited for low contention data scenario and situation where locks are not possible which models the web/distributed world pretty nicely

  OCC involve following stages;

  Begin – start of transaction
  Modify – fetch records, modify, add, delete
  Validate – validate that the affected data are not intersecting with other transaction
  Commit/abort – make the changes permanent if validate is OK else abort

  It is assumed that the db was opened with the proper flag to allow transaction to happen. As of now only occ is supported hence user should open db as follows (in case of embeddable db)

  database *db = new database("mydb", DB_OPTIMISTIC_TRANSACTION);

  And in case of server, just enable the transaction in the bangdb.config

  DB_OPTIMISTIC_TRANSACTION = 1

- ***APIs***

  BangDB exposes following simple set of APIs to use transaction

  void *begin_transaction();

  returns a opaque transaction handle to the user which should be used for all the operations used in the transaction boundary.

  long long commit_transaction(void **txn);

  returns the transaction id when successful else -1 on error. It does the validation and if OK to proceed then commits the transaction, which will make the records permanent. If validation fails then it aborts the transaction.

  Void abort_transaction(void **txn);

  Explicit method to abort the transaction.

Copyright © 2014 IQLECT All rights reserved

Note: The function also releases and cleans up the various data structure involved with the transaction including the transaction handle (void *), hence user is not required to do anything else beyond this. If commit fails (return value less than 0) then user is free to retry the transaction starting with getting the new transaction handle by calling the begin_transactions()

- ***Autocommit***

  This is enabled by default, which means that every single op will have implicit ACID support even if transaction is disabled. But when autocommit is disabled and transaction is on then user will have to use the transaction explicitly (begin and commit) even for a single operation (read or write)

- ***Atomicity***

  Each transaction is treated as a single unit of work. Hence either all succeed or all fail. BangDB ensures this by applying the 3 steps parallel validation to ensure serializability

- ***Isolation***

  Each transaction is isolated from the other concurrent transactions. Hence the changes done by a particular transaction is not visible by other transactions. Since BangDB implements the highest order of isolation hence following are **not** possible

  - Dirty Read
  - Non-Repeatable Read
  - Phantom Read

- ***Consistency***

  Concurrent user transactions never leave the db in in-consistent state. Since validation is done before committing and transactions are rolled back when issues are seen thus ensuring that only consistent transactions commit and apply changes to the db. Also user never see inconsistent state of the db as each transaction is isolated from other one fully

- ***Durable***

  Once the changes are committed, they become permanent. WAL helps in this regard as logs are flushed to the disk which is enough to ensure the durability as it can be replayed to get the db state back to normal in case of db crash or abrupt process termination

Copyright © 2014 IQLECT All rights reserved

4. Performance

Contrary to general belief, the transaction don't bring down the performance too drastically if implemented in right fashion. The one of the major reasons for implementing OCC was the performance as it allows the concurrency of highest level to proceed. Also the parallel validation further improves the performance. For BangDB here sis the quick comparison of performance

Index	Without Transaction, Log – ON		With Transaction, Log - ON	
	Write (ops/sec)	Read (ops/sec)	Write (ops/sec)	Read (ops/sec)
Btree	475,000	1,025,000	250,000	800,000
Hash	500,000	1,690,000	275,000	875,000

Please check out the website for more details on how to use transaction in BangDB

Copyright © 2014 IQLECT All rights reserved