BangDB – Feature Spec
Network DB

www.iqlect.com

1. About

BangDB Network version is pure vanilla key value nosql client server data store with master slaves cluster model. The goal of BangDB is to be fast, reliable, robust, scalable and easy to use data store for various data management services required by applications. In this flavor, db runs as network service and clients access it over the network. This model is good for sharing data with multiple apps or instances of an app. The typical use case for this flavor is cache on top of the database, a network data store etc...

2. Feature Spec

Following are few high level feature set for the BangDB Network DB which will be supported in the first release;

(a) Shared Server

BangDB runs as service and clients connect to it over the network. The same db instance is shared with all the clients and clients can connect to the db in concurrent ad-hoc fashion. The DB ensures that the clients are served in efficient and predictable fashion

(b) Persistent and In-memory

The BangDB server can be configured to either run entirely in-memory or run as a data store. When used as data store, one can enable write ahead logging (WAL in default mode is disabled) and set the flush frequency based on their need. The replication can also be enabled to ensure switching from one machine to other when required

BangDB can be configured to be used just as cache or as persistent store. On one hand, the user can configure bang db to have all the data in memory and not to go to disk for any operation. But while closing the db user may decide to take the snapshot of the data in memory and store it on the disk for later use. And on the other hand, user can start up-front with the disk backing where data will overflow from memory to disk when required. The data is consistently and frequently being flushed to disk to clear up free space in the buffer pool. The continuous and sequential log flush ensures data durability even though the data itself is not written to disk most of the time until needed

For a corner use case where data durability is of utmost importance and performance not as critical, then user can run db in a mode where it directly writes to the files on the disk, with the same btree and ext hash implementation. This option allows multiple processes to operate on the same db simultaneously. Typically to store extremely critical, highly durable but small amount of data, this option could be used

(c) Robust, crash proof, available, resilient

BangDB core implements write ahead log and offer it as part of configuration for the db. When data durability is required, user should enable log and set the log flush frequency as per need. The db takes care of frequent write of the log to disk, hence in the event of data not being written to disk and db crash, BangDB recovers the data when restarted. BangDB frequently check points the log as well in order to speedup the data recovery process by replaying the log in case of db/machine crash.

The db provides the least granularity compared to many other dbs for durability of the data. For ex; user can set the log flush frequency in Milli seconds, this is critical as in case of process crash or any other such events, data loss would be minimal as db would try to recover data as much as possible. BangDB implements the variant of ARIES algorithm for write ahead log for data durability and at the same time having least impact on the performance.

(d) Replication

The replication of data will be enabled by default. User can opt for no replication as well by setting the configuration appropriately but in NetworkDB scenario it is recommended to set the data replication ON. User can also go for log replication (sync or Async) apart from data replication. The data replication, especially in sync mode allows user to switch between master and standby at run time

A master can have many standbys and each can communicate with other. The synchronization of standby with master in non interfering. User may chose to read from slave and write to master as required

Replication of data with secondaries or slaves can happen at run time. Which means that while master is still serving the data to multiple clients, many slaves together can come and sync data with the master. Master sends various files and information to slaves in order to make them up to date. Slaves on the other hand can quickly become available for clients even though part of the replication might still be in progress. This design improves the performance and availability of master and slaves even during sync process

Master and slaves checks each other health using light weight udp based ping protocol. When master notices that slave has been disconnected or unavailable for some reason then it removes it from it's slave pool eventually. Slave on the other hand can keep on trying to connect to master but eventually it will also give up if master is unreachable. Finally new master has to be selected among slaves and that node takes over from thereon.

(e) High performance

One of the most important design goals for the BangDB is the performance. The design and implementation from scratch have helped to learn from other's experience, leverage new concepts for the high performance. BangDB in both embedded and Network flavor is achieves very high performance even when compared to other very high performance DBs in the market, for ex; Redis. Separate document will cover the performance comparison of BangDB with Redis in several scenarios

(f) Highly Concurrent

Concurrency in BangDB cuts across all the layers. It's designed to leverage multiple cores on the machine by allowing multiple threads to execute in parallel with efficient locking mechanism. But user can select the number of threads that should execute in parallel.

On the other hand, the server is highly concurrent and is capable of handling thousands of connections at a given time with very less overhead. BangDB implements epoll in Edge triggered manner to handle client requests with efficient staged driven architecture to perform actions on each requests in asynchronous manner. The BangDB follows staged event driven architecture to create well conditioned, scalable and available server which can tolerate very high number of concurrent connections and continue serving requests in expected manner even in an extremely loaded scenario, which is very difficult to achieve using the conventional design and architecture and requires huge amount of continued effort and cost to even survive business in the face of higher and unscheduled spikes of load

The BangDB client runtime and server enable user to create data centric, well conditioned scalable network application focusing totally on the application logic without worrying about the underlying complexities of network communications, protocols, error handling, sustaining very high concurrent connections (C10K Problem), high load and other related issues for data related activities. In the event of very high concurrent connections, unlike other services which in many cases just stops responding to the clients, BangDB degrades gracefully for all the clients instead of just denying services to few while serving other few

(g) Runs on commodity hardware

BangDB is designed to run on commodity hardware. User may add SSD for performance reason especially for storing log files on the SSD. But in typical scenario this is not required as BangDB gives very high performance on a typical commodity hardware

(h) API

BangDB supports both thick and thin client. First version will support only thick client where every client will have to link to a lib in order to talk to server. Though the API for thick client will be same as of embedded or cluster (future) version to ensure same code runs in all cases.
Future release of BangDB Network version will support client connecting using HTTP for GET, PUT DELETE operations

(i) Consistent API, Easy to use, deploy and manage

All flavors of BangDB have the same API. Basically the flavor of the BangDB is in a way abstracted from client. Client always sees a single BangDB to operate with. It's beneath the single machine view of the BangDB where we may find entire cluster of nodes. This simplifies the code porting in case one wants to move from one flavor to other. Since BangDB provides the same old get,put,delete API hence it's pretty straight forward for developers to start playing with the db. The overall integration time is very less depending upon the application's state.

However, an admin portal will be provided in second release of the Network DB to handle everything as far as server/standby/log/replication/snapshot/monitoring/ etc.. are concerned. The portal will also render the real time snapshot of the various data and statistics in graphical user interface

(j) Access Methods

BTree and Hash options for access methods for data will be supported in the NetworkDB. User can pick one of these based on their requirements by setting the appropriate flag in the configuration file

(k) Transaction

For every single operation, BangDB ensures that either all parts of db is updated appropriately or the operation fails. This in turn ensures that incomplete operation (due to any reasons, machine crash, server crash etc...) would never leave DB in inconsistent state

Multi operations transaction is also supported in the DB. BangDB implements optimistic concurrency control to achieve very high number of concurrent transactions to proceed while guaranteeing full ACID support

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)

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

(l) Configurable

BangDB has got only one configuration file namely bangdb.config and user can configure many aspect of the DB using this config file. Some of the key configuration are as mentioned below;

- PAGE_SIZE – user can set different page size for the db based on usage requirements
- BANGDB_DB_TYPE – set the db type, whether in-memory or persistent etc...
- BANGDB_INDEX_TYPE – set the access method
- BANGDB_LOG – set log ON or OFF
- DAT_SIZE – max data size that user can set or get in single call, default 2 MB
- IDX_SIZE – the key size (min 8 bytes, max – no theoretical limit but limited by page size)
- MAX_THREADS – how many concurrent threads client can use for server interaction
- SERVER_ID – ip address or server name
- SERV_PORT – port num
- BUFF_POOL_SIZE_HINT – buffer pool size, hint because DB might change (less than 0.1%) it
- LOG_BUFF_SIZE – Log buffer size
- LOG_FLUSH_FREQ – the frequency of log flush in micro sec
- CHKPNT_FREQ – check point freq in micro sec

and many more... please see the config white paper for detail information

Similarly, the server initializing code also has few configurable parameters such as number of stages, number of threads in each stage, handler etc... Though for BangDB Network version, user will not have to do anything over here expect setting the mentioned configurable parameters. But in future the building blocks will be made available for user to write their own network services

(m) Message Protocol

BangDB implements its own message protocol to send and receive messages. BangDB has implemented its own RPC mechanism with custom message format, Marshalling and unmarshalling mechanisms. This has been done to ensure the performance and robustness of the various components in different scenarios

Beneath the layer, BangDB uses the iqlectEQ which is scalable distributed event driven messaging platform, which will be released later next year for user to build their own network applications without worrying about the inherent complexities of network communication, distribution issues, performance and scalability, high availability of the services etc... The iqlectEQ can be used to create an application by loosely connecting many distributed components through the iqlectEQ messaging platform. The concept of stages are used to split or divide an application into multiple stages and then connecting them through the iqlectEQ platform to create a scalable, available application using the commodity hardware

More on iqlectEQ will come in separate documentation

(n) Memory Management

BangDB sever does various kinds of memory management. Apart for the typical buffer pool for the BangDB, it also manages all the memory required for various activities by the server. This ensures that server never really runs out of memory, instead when in pressure it just waits until memory is available which even in highly loaded and stressed scenario should not be high. This also ensures that server never rejects client requests due to memory pressure but it just makes them wait until memory is available (wait time is very small even in stressed scenario). Client can in turn apply the wait time for request to complete which ensures that client doesn't wait forever. This also helps improve the overall performance

(o) Time Out

User can set time out for each connection for various operations. Server will take appropriate steps to comply with the time out set by the user

3. Release

BangDB Network version has been released under the BSD license as free of cost