BangDB – Product White Paper

www.iqlect.com

1. About

BangDB is a key value nosql data store. 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. Few highlights of the db are;

BangDB comes in many flavors

The BangDB can be used without changing the logic or code in many fashions suitable for different needs. BangDB is easy to use db as part of process, as network store or in the conventional load-balanced clusters. Following are the supported flavors;

Embedded (In Proc) In memory

The db becomes part of the process (hence embedded) and provides db services to the application directly. This flavor is good for apps that want to access data in fastest possible manner, without incurring network overhead. The typical use cases for this db are for ex; caching product catalog at the application level, storing semi static data near the application, app specific data, local computational data etc...

Network Server

The db runs as network service and clients access it over the network. It is basically a client server model where user can create the cluster of nodes in which one is master and rest are slaves or secondaries. 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...

In memory, distributed data grid/ Elastic Cache

This model is single machine view of the entire data cluster where each node runs an instance of BangDB. The nodes (machines) are dynamically added and removed from the cluster / grid at the run time as and when required with high churning rate of the size of the cluster. The data is distributed across the cluster and individual node provides the services for fraction of the overall data. Typical use case for this model would be to hide the relational database from the apps and provide the data management services from the layer itself. This cuts down the pressure on the RDMBS and hence provides multi fold linear scalability to the application from the data management perspective. Other use case would be to provide shared distributed data cache for applications for performance, throughput, scalability, high availability etc..

This option is also very attractive for dealing with BigData. Since BigData mostly come in real tine and with high volume and velocity. This model can be used for data collection and store for BigData as the high performance of the DB along with ealstic in nature would provided the much needed flexibility and enable the app or user to deal with BigData in real time

Persistent data store (not just caches)

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

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 (theoratically in micro sec), 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 write ahead log for data durability and at the same time having least impact on the performance. it is important to note that the db performance is exceedingly well and best in most conservative setting of the db parameters for data durability. It will be covered in detail in the technical detail section. The user can opt for enabling or disabling log, in-memory only, etc... by setting the right configuration

For a corner use case where data durability is of utmost importance, user can run db in a mode where it directly writes to the files on the disk, very similar to dbm but 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

Scales to available memory, elastic in nature, throw machines to scale linearly

Scaling BangDB is easy, Just add more machines. With the elastic data grid, user will just have to throw in machines to scale to new load linearly. By adding more machines or nodes, the other nodes in the cluster adjust accordingly and start allowing newly added node to take load instantly for increased throughput and lower latency without reconfiguring the whole data sets across the cluster

The need of cost consciousness in the design is critical and hence purposing, re-purposing, provisioning, growing and shrinking have to be done efficiently and without affecting the overall system. Hence the BangDB elastic cache is designed to tolerate high internal churing

The design of BangDB leverages the available and allocated memory to the fullest. In-fact reserving more memory for the db ensures better performance for high volume of data. A typical node having BangDB can store theoretically terabytes of data, but amount of memory would decide the practical limit. However, to handle even bigger data (in multiple terrabytes or more), adding more machines would help. It is important to note that the distributed data grid or elastic cache implementation allow user to just throw machines to scale linearly to handle more load. The one machine view of the grid allows user to treat a embedded, single instance network or whole data grind in similar fashion. Clients are basically unaware of presence of data grid in the system. The peer to peer implementation of the cluster topology helps having a system which is self monitored and managed with least impact in the event of churning

High performance, high concurrency

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. The BangDB boasts of being the fastest in the market in its own category as of now. The comparison section would elaborate more on this in detail. The BangDB is highly concurrent and runs parallel operations as much as possible. For ex; the hash implementation of index is lock free for reads, similarly the Btree implementation takes on an average less than two locks on the Btree pages for a write operation at any given time. The various design techniques in write ahead logging, buffer pool design and the background workers have allowed the BangDB to achieve very high performance with less amount of code. As of now as per our benchmark analysis, BangDB runs faster than Oracle's berkley db and Google's leveldb

In the virtualized scenario, the emphasis on one to one mapping from the performance and resource consuption has become accentuated. The X% performance increase could mean upto X% of less hardware resource consumption and also less overhead on operational cost. Hence BangDB in this regard becomes an attractive option for organizations who want to migrate their applications to cloud for less capex and opex

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 replication of data/log will be enabled in the elastic cache flavor. The elastic cache ensures that the db is available as a cluster to the user at all times even though nodes are leaving or joining the system. For ex; after writing a data to a node if node fails, the system still allows user to retrieve the data transparently. The BangDB cluster ensures that the it adjusts at run time in the event of nodes failing, joining or leaving the system with higher churn rate.

Runs on commodity hardware

BangDB is designed to run on commodity hardware. It can run even with smallest amount of memory committed to it. For ex; it has been tested with even less than 1MB allocated to it for buffer pool for over a Million reads and writes. But that's for the test purpose only, in real practical world, user may allocate as much memory as needed or available. BangDB scales with memory. BangDB is also a concurrent db which requires minimum 2 CPU machine to run in multi-threaded scenario. All the performance evaluations are done on commodity hardware only. With better and powerful hardware BangDB will perform better, but the numbers indicate that commodity hardware should be more than sufficient for almost all use case scenarios

Transactional

BangDB can run in transactional or non-transactional mode. This can be set on or off using the bangdb config file. When transaction is off, user is not required to wrap the operations in begin and commit transactional boundary, instead simple direct call is enough. Though every single operation supports full ACID but for wrapping up multiple operations in side a transaction, user will need an explicit transactional boundary. Hence for multi ops transactions, user can open db in transactional mode and can carry on operations inside the begin and commit boundary.

BangDB currently implements optimistic concurrency control for transactions to ensure full ACID. The db uses parallel serializability to provide highest order of isolation. BangDB maintains the high level of performance even in the concurrent transactions mode by ensuring the it continues to leverage the number of CPUs on the machine.

Consistent vs available

The write to a particular node is always consistent. In the event of some issues faced mid way of a write operation, BangDB will undo the operation to enforce ACID concept. However across the cluster user has more than one option

BangDB can be set as ACID within a node of the cluster and eventually consistent across the cluster. User has the option of setting the configuration in terms of degree of non-consistent view of data that can be tolerated. BangDB can also be set to give consistent view across the cluster in future version of the Elastic Cache

Shared nothing and self managed p2p based cluster

The p2p based fabric ensures that the cluster is self monitored and managed. The lack of master slave kind of design ensures that the single point of failure or bottleneck is avoided hence to gain further performance improvement and robustness

The nodes and the keys in the cluster are identified as SHA 160 bit value. All the nodes or machines in the system are assigned a unique 160bit value on an imaginary ring to leverage the consistent hashing approach which in a way allows the cluster to have high churning in terms of nodes leaving and joining without affecting the SLA

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. If application is already using some kind of key value store or OR mapping with the db, then it's a matter of few minutes to hours to complete the integration. On the other hand if application wants to have a local cache or store apart from the existing relational database, it's again very easy and fast to integrate. When application wants to move away from relational database and if it was using the sql queries inside application, then it will take time in the tune of weeks to complete the integration with BangDB

BangDB is easy to manage as it doesn't require any db admin. All flavors are self managed. The elastic cache will have admin portal though to view some stats and health of cluster. Also user can take some action using the portal as needed. But in no way it demands an admin to be allocated for the BangDB.
The deployment of BangDB single instance is just copying library. For upgrade it's again the same as long as the new version doesn't change the data format on the files

NOTE: Some of the above features would be available in coming network db and elastic data cache/grid releases. The current release is for embedded nosqldb

2. Feature Spec

The high level feature description is given below;

a) Embedded (In Proc) In memory

1. key value store
2. in-memory only and/or persistent
3. highly concurrent
4. buffer pool, exploits temoral locality
5. asynchronous write to disk
6. B+link Tree and Ext Hash index
7. write ahead log
8. crash recovery
9. very high performance
10. Transactional (OCC)
11. multi table types – Normal, Wide and Primitive
12. multi indexing
13. json data support – nested index
14. composite key

b) In memory, Network

All of the above and;

1. poll based event handling

2. message coordinator
3. staged event driven

c) In memory, distributed data store/ Elastic Cache

All of the above and;

1. distributed fabric
2. p2p based structure
3. consistent hashing
4. SHA based 160 bit keys
5. replication
6. CAP knob (consistency vs availability)
7. provisioning, deployment
8. failure detection, monitoring
9. no admin and management
10. node auto reconfiguration
11. and more....

(d) Abstraction for data analysis

1. Sliding Window – slides with time for real time data analysis
2. Counting – Absolute, probabilistic, unique, non-unique
3. TopK – topk items
4. get and put API for all analysis, all complexities inside abstraction

Note: The details on (b), (c) & (d) would be covered in detail in their separate doc

4. Usage Scenario

The typical use cases for BangDB are no different than other nosqldbs in the same category. For ex; for BangDB as embedded db, the use cases for berkleydb or leveldb will also be the use cases for BangDB to great extent. However, the significant performance gain with BangDB (see the performance document) makes it attractive for apps requiring higher performance. With the app virtualization and infrastructure as service, higher performance would somewhat mean to lower in capex and opex as the required number of operations can be achieved with relatively less resources.

To jot down few, here are some of the scenarios in which using BangDB would make sense;

1. When relational database becomes bottleneck and refuses to scale with increased traffic at acceptable cost
2. For storing quasi static data at the application level itself for improved performance
3. For storing shared data in level above relational database for scalability, for ex; session related data
4. When business generates lots of temporary or local data which may not belong to the main data store. Ex; shopping cart, search caching, site personalization, partial form data etc...
5. When relational db has been normalized for performance
6. When dataset contains large objects, text or binary
7. To allow application to scale linearly by partitioning and distributing data across multiple nodes
8. To handle high cluster churning
9. For high availability and failure tolerance
10. many more ...

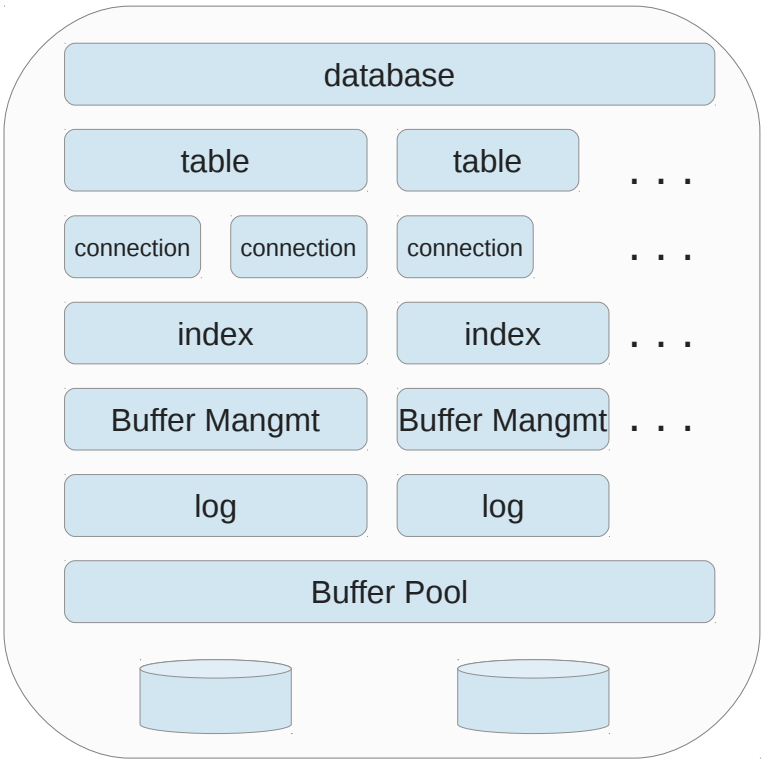
The iqlect is also working on a plug in component to enable real time view on the BigData. It will be off-the shelf component to be plugged in to the analytics infrastructure which will allos user to collect data coming in with huge velocity and volume and at the same time allow user to do eventing and some basic analysis on the data to

get the sense of it in real time. The structure also complement the hadoop ecosystem by sitting before the hadoop interface and allow hadoop to work more efficiently

5. Architecture

The BangDB provides simple standard APIs for clients to access the database. Please see the API section for detail. When enabled, BangDB creates a buffer pool of size given by the user and then creates many data structures to mange the buffer. It creates a hash table of buffer headers, an lru list, a dirty page list and a free header list. It also creates workers to handle the various housekeeping for buffer pool and also for flushing dirty pages to disk asynchronously. The BangDB takes decision at regular interval depending upon the pressure on the memory and requirement to decide on how much buffer should be freed, how many pages to be flushed etc. The lru list helps in deciding which headers to be flushed before others

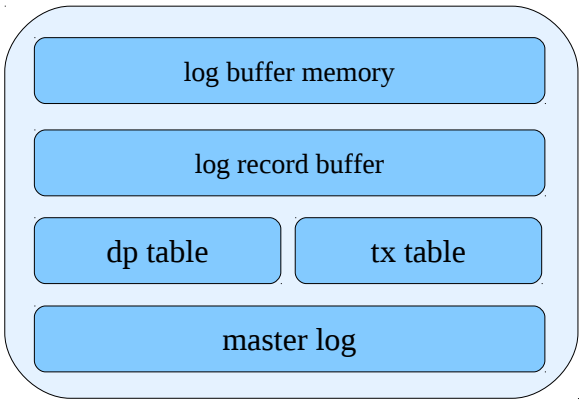
Logically the BangDB consists of mainly following components as in simple view given below;\



The user creates the index again based on the configuration. Btree can be used when balanced read and write is needed and range of data could be of some importance or else ext hash based index may be used otherwise for similar write(as with Btree) but much higher read performance. The index deals with buffer pool for all it's data requirement. When data is present in the buffer pool, it gets it from the pool itself (a cache hit) else buffer pool reads it from the disk and then returns to the index

When enabled, write ahead log(wal) keeps on writing the individual operations in the log file and keeps on rotating the buffer as and when it gets filled. The wal provides the log check-pointing and replay functionality which helps in recovering data when db was not closed properly or in case db/machine crashed. There are multiple workers for wal which keeps on checking to do various housekeeping jobs. One of the workers wakes up regularly and flushes log if required. The log is flushed in bulk and being sequential write it happens relatively quickly. All log flushes are synchronous hence gurantee the persistence of log data which is critical.

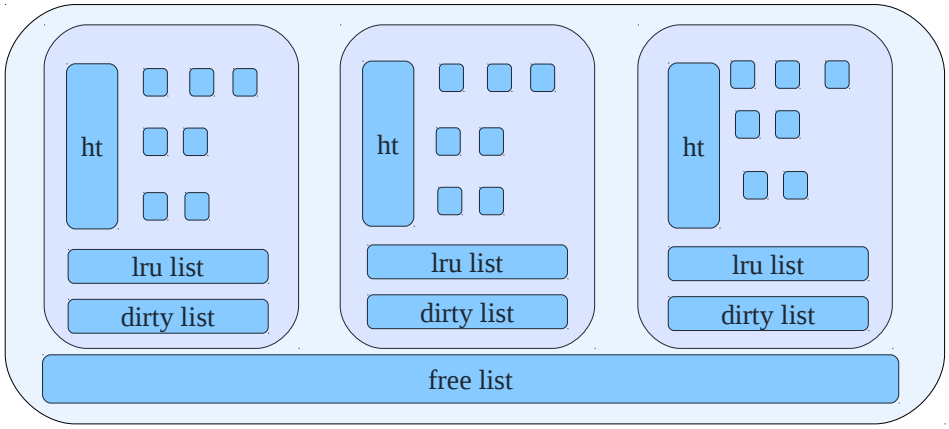
The write ahead log(wal) contains log shared memory, record headers, log records, master log various handles etc. The high level view of the wal is as following;



User creates a database type and then request for a table, and then using table user gets the connection. The reason for database and table indirection is to have scope for future implementation of mulitple tables within a database. The connection is the handle using which user interacts with the data. Note that the user may create as many connection she wishes (limited by the max connection defined in the config file) and pass them on to different workers for use. The stat collection using different connections could be a motivation for the same or let individual thread or workers create connection when required and get rid of them once done

The index as shown above is basically implementaion of access methods for data. These access mechanisms are implemented as B+link Tree and Extended Hash. The use of one over other purely dependent on the context and choice of user as it's available as configurable parameter

The buffer pool consists of headers hash table, lru list, dirty page list and free list. This is how the buffer pool would look like for ext hash as index type, for ex;



Please note that BangDB creates separate pool for different file type. This is mainly because db maintains different files for index, data, dir etc... and hence keeping separate pool encourages more concurrency for working on the buffer area. We can further fine tune the flushing of dirty pages or reclaiming free pages depending upon the nature and criticality of the data. For ex; for Ehash type, dir file size is quite low hence entire data can always be kept in memory without allowing flushing of dirty pages or reclaiming. The free list however is common and it indicates that all has equal rights to get free page when in pressure

Note: The architecture for networkdb and elastic cache/grid would be covered in separate documents.

6. Performance

The purpose of the performance analysis of BangDB under few scenarios is to present a high level mesurement figure which may help users to easily map their use cases and understand what to expect from BangDB. The performance measurement is done on commodity hardware without doing any customization.

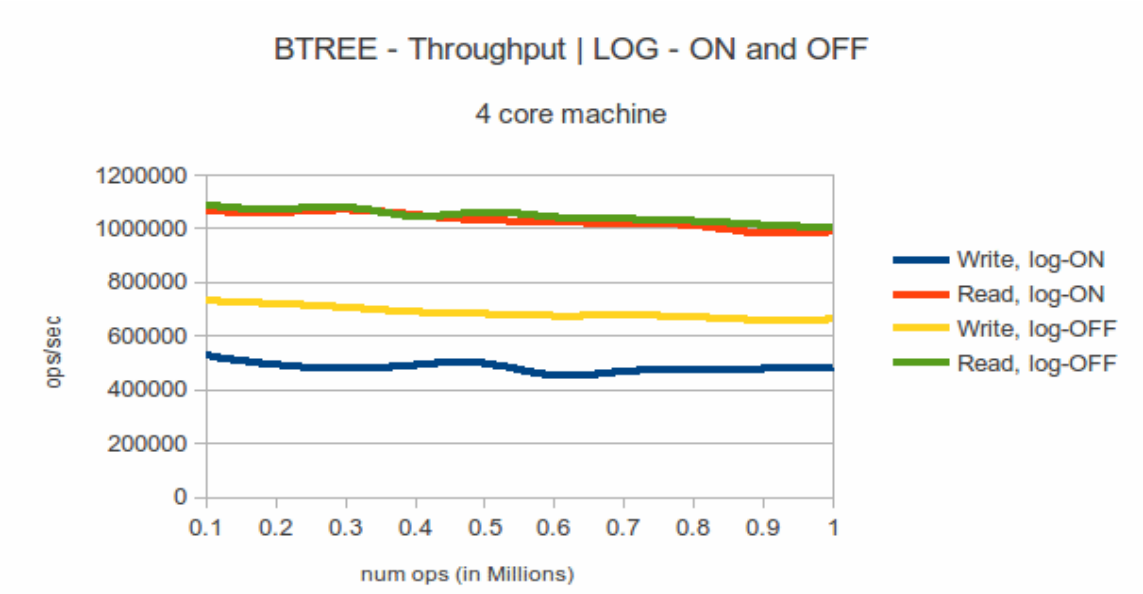
These performance numbers will vary depdending upon the configuration of machine, OS, size of key and value and other parameters, hence users may see different metrics when they run on their own machines in different settings. However, the benhcmark shown here would help user to take decision in some fashion.

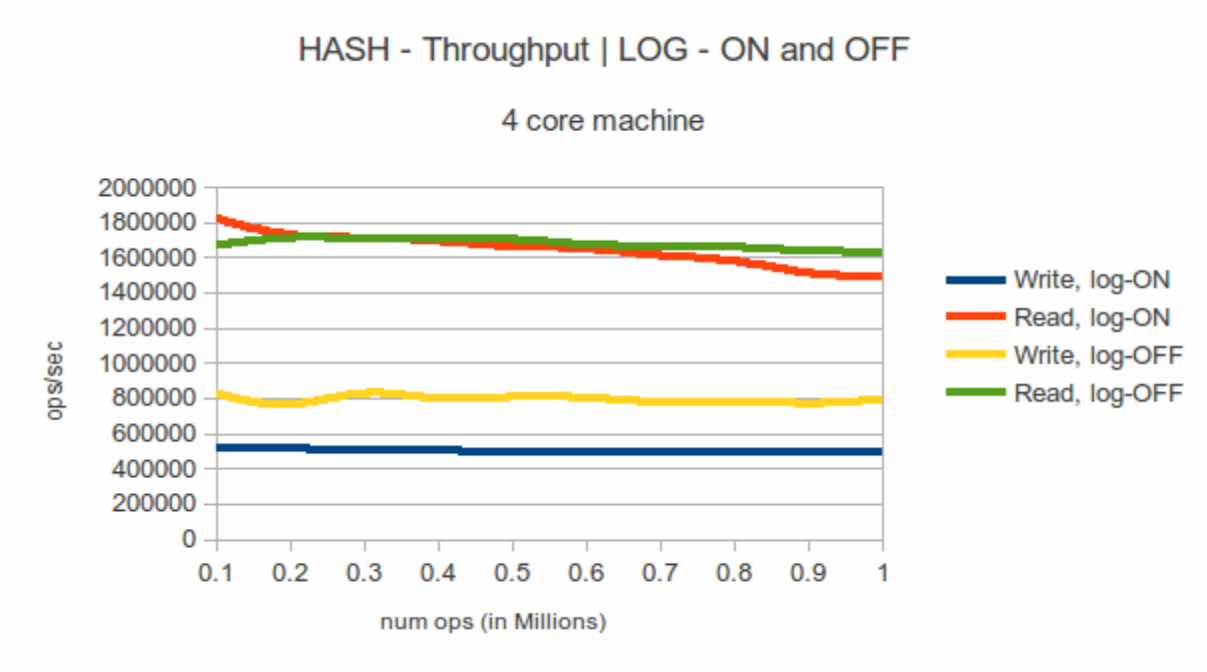
This configuration ensures that the db runs in conservative mode where if process or db crash happens, at restart the db will recover to the point where the db crashed. There are many workers who are ensuring that the mechanisms for write ahead logging and buffer pool health. Note that in the table the data with the Log = ON depict the numbers for above configuration. However if we switch off the log and just work with everything else as it is(apart from log and related stuff) then the numbers would look like as given in the column for Log = OFF.

The average values for the throughput(ops/sec) over the 100K – 1M operations are;

Index	Log – ON		Log - OFF	
	Write (ops/sec)	Read (ops/sec)	Write (ops/sec)	Read (ops/sec)
Btree	475,000	1,025,000	685,000	1,045,000
Hash	500,000	1,690,000	790,000	1,675,000

The following figure shows how the operations per second fares with the number of operations. A million IOPS would write around 1 GB data including log files in this case





In the above figure the writes are consistent for both btree and hash and the gaps between the 'log' and 'nolog' is due to the overhead of writing log record before the data in-memory, and other background jobs done for write ahead logging for ex; log flush, log split, log checkpointing etc...

The reads are shown for various number of operations in the above figure. Note that the numbers fo reads for both log ON or OFF are very close and similar, the reason for this is because for read there is no logging done and hence no overhead of logging despite the status of log(ON or OFF)

7. Getting started

Please log into www.iqlect.com and download the BangDB binary and the getting started document. Please see the ReadMe for installation info that comes along with the download.

There are many other documents are available at the site to get the detail insight of the db.