LA-UR-22-28759

Approved for public release; distribution is unlimited.

Title:	API Requirement for LANL's Next-Gen KV-Based Storage
Author(s):	Zheng, Qing Manno, Dominic Anthony
Intended for:	Report
Issued:	2022-08-19 (Draft)

API Requirement for LANL's Next-Gen KV-Based Storage (Draft, Ver1.0)			
1	**Device-Level Operations**		
	API	Description	Notes
1.1	KV_dev_format	Format a given device. Reset to factory. Purge all existing data. Clean up all error states.	A device can be identifyed by an address string using a format determined by the vendor.
1.2	KV_dev_ping	Ping a device.	The goal is to check device existence over a network. It can also be used to verify if an address has the right format.
1.3	KV_dev_open	Open a device and return a handle to it for followup operations. A client may be required to specify certain configurations.	A device should allow multiple processs and threads running on potentially different compute nodes to concurrently open a device and simultaneously perform operations on it.
1.4	KV_dev_open_readonly	For better perfomance, security, and potentially less in-memory state mangement, a device may be opened with only read accesses.	Readonly objects are always easier to work with.
1.5	KV_dev_stat	Report device level stats such as current logical space usage, physical space usage, space left, current keyspace count, and potentially other per-device information.	This will be something similar to fsstat.
1.6	KV_dev_close	Disconnect from a device, release client side resources, and release the device handle.	A client may abort a program without calling KV_dev_close. For example, a program terminated by Ctrl+C.
2	**Keyspace-Level Operations (in general all keyspace operations require a dev handle)**		
	API	Description	Notes
2.1	KV_keyspace_list	List all existing keyspaces within a given device.	Keyspaces are identified by names. Keyspace names are unique within a device. Implementation is expected to allow at least 255 characters for a keyspace name (similar to a filesystem's NAME_MAX). Keyspace names are not necessarily C-style strings.
2.2	KV_keyspace_exist	Check existence of a certain keyspace.	
2.3	KV_keyspace_create	Create a new keyspace with a user-specified keyspace name. If a keyspace with the given name already exists, either return an error (O_EXCL) or remove all existing data within the keyspace (O_TRUNC). A client may be required to specify certain configurations for the keyspace.	
2.4	KV_keyspace_open	Open a keyspace and return a handle to it for followup operations. If the given keyspace does not exist, either return an error or dynamically create it (O_CREAT).	Multiple processes and threads from different nodes may simultaneously access a single keyspace.
2.5	KV_keyspace_open_readonly	Open a keyspace with only read accesses to ease concurrency control and state management.	
2.6	KV_keyspace_compact	Seal a keyspace (no more writes) and request compaction on it. In general, compaction sorts data by key and creates an index on the keys. In addition, compaction also builds a histogram on the keys so that a client can later retrieve the histogram and know the key's distribution.	The idea is that a writer creates a keyspace, opens it, inserts data into it, and calls compaction to have data sorted and indexed. Then, a reader opens the keyspace and performs queries. In general, scientific simulations tend not to read their data until after the simulation is done and all data is written to storage.
2.7	KV_keyspace_is_compacted	Check if a given keyspace has finished compaction and is ready for queries.	
2.8	KV_keyspace_histogram	Return the histogram built by the compaction process.	
2.9	KV_keyspace_stat	Report keyspace level stats such as current logical/physical space usage, current key count, and other per-keyspace information.	
2.10	KV_keyspace_close	Close a given keyspace, release client side resources, and release the keyspace handle.	A client may abort a program without ever calling KV_keyspace_close.
2.11	KV_keyspace_delete	Delete a keyspace and all its data.	Deletion may be deferred when there are one or more outstanding handles to the keyspace, in which case the keyspace will be deleted when all parties release their handles.
2.12	KV_keyspace_export	Export data from a keyspace to a file.	Data may be exported into a user specified file format such as SSTable, Parquet, HDF5, or other open-source formats.
2.13	KV_keyspace_import	Import data into a keyspace from an external file.	
3	**KV-Level Operations (in general all KV operations require a keyspace handle)**		
	API	Description	Notes
3.1	KV_kv_put	Insert a single KV pair into a given keyspace.	Clients can ensure that keys are unique (no conflict). For a given keyspace, KV sizes are always fixed (no varlen K or V is required). Implementation should support keys up to 256B and values up to 4KB. Data persistence at keyspace level is sufficient (KV-level persistence is not required, either an entire keyspace is retained or lost after all keys are inserted).
3.2	KV_kv_bulkput	Insert a batch of KV pairs into a given keyspace.	
3.3	KV_kv_get	Point query on a key.	Keyspace must have already been compacted before queries may take place.
3.4	KV_kv_rangeget	Range query over keys.	Clients provide ranges in the form of min and max.
4	**Advanced Query Operations (multi-dimensional query capability)**		
	API	Description	Notes
4.1	KV_query_create	Create a query object from a user supplied SQL-like query string such as "select X, Y, Z where E > 3".	KV schema (which portion of a Value is X and what is its data type) can be specifed at keyspace creation time. For example, byte 0-3 of V is X and X is a float.
4.2	KV_query_estimate	Estimate the result for running a given query on a specified keyspace.	This is to use keyspace indexes to estimate the number of KV pairs that might match a given query. This helps a client prepare a large enough buffer space to host the actual query result.
4.3	KV_query_run	Execute a given query on a specified keyspace.	
4.4	KV_query_release	Release the query object.	