**Schema Modification Proposal**

This modification proposal focuses more on permanent data schema, since the string basic data type in Redis is fairly good for temporary data since temporary data can be frequent I/O without any pattern. String data type is small, low I/O cost and versatile, hence I agree with using String datatype for temporary datas. However, as for permanent data, we need to consider the integrity, memory cost and also data fragment in long term. To solve issues above, I propose few changes to permanent data schema as solution.

**User information & sensor information (permanent**):

Permanent data is separated in 3 sections: user information, sensor information and user-sensor association information. The original schema is based mainly Redis string data type and saved as pairs. There are few issues. For example, for user information, it is difficult to tell total profile validity, such as missing information. The only way to retrieve the list of all profiles is through “keys” command which involves numerous keys retrieval which is i/o costly. Hence I propose using hash-list method for user information and sensor information. Refer to excel for specific schema stable.

**Operation implementation (use sorted set to implement list feature):**

**Add**:

*e.g: add sensor with ssn=00001 with field wmac, cmac and stat*

HSET s:info:00001 wmac 127.0.0.0 cmac 127.0.0.1 stat 1

ZADD s:list 1 00001

**Delete**:

*e.g: delete the entry added above*

DEL s:info:00001

ZREM s:list 00001

**Search**:

*e.g: find sensor with stat of 1*

for i in sonsor index:

check HGET s:info:i stat

return if stat==1

*e.g: check existence of sensor 00001*

SSCAN s:list 0 MATCH 00001

*Note: do not use KEYS 00001 since it is a blocking operation. We want unblocking operation for DB requests response.*

**Sort**:

*e.g: sort sensor cmac by wmac.*

SORT s:list BY s:info:\*->wmac GET s:info:\*->cmac

**User-Sensor association information (permanent):**

Since this table is also permanent, we do not want large amount of keys. Using String for association information can create quit large amount of keys. For example, if we have n users and m sensors with k different configuration, we will create n\*m\*k number of keys, which is difficult to track and it is special costly. To better resolve it, I propose using 2 lists to track mapping information from user to sensor and sensor to user. Also use hash to track specific configuration between user and sensor. With the same previous example, the special cost would be 2+n\*m at maximum.

**Operation implementation:**

**Add**:

*e.g: associate sensor (ssn=001) to user (usn=jack) with configuration (color=black)*

SADD u\_s:mapping:jack 001

SADD s\_u:mapping:001 jack

HSET u\_s:config:jack:001 color black

**Delete**:

e.g: dissociate above relationship

SREM u\_s:mapping:jack 001

SREM u\_s:mapping:001 jack

*Nots: the configuration is preserved by default in case relationship is restored in the future. If specified to remove configuration:*

HDEL u\_s:config:jack:001 color

*To remove whole setting*:

DEL u\_s:config:jack:001

**Find**:

*e.g: find if 001 associate with jack*

SSCAN u\_s:mapping:jack 0 MATCH 001

SSCAN s\_u:mapping:001 0 MATCH jack

*Note: this is double checking the association, which improves data integrity*

*e.g: show all association of jack and sensor 001:*

SMEMBERS u\_s:mapping:jack

SMEMBERS s\_u:mapping:001

**Sort**:

*(function not needed. If this function is preferred, change implementation to sorted set instead of set)*

**Measured air raw data (temporary**):

Sorted set is a great optional. However, we can take better advantage of new data type STREAM. STREAM has built in timestamp, which we do not need to worry about lock sync cross all sensor. Instead, time will be synchronized without Redis DB clock internally. The STREAM also has built in blocking operation, which is perfect for listening and data streaming. This is very trivial change since sorted set and stream shares most properties.

**Add**:

*e.g: add sensor 001 data stream with data CO2=10*

XADD data:sensor:001 \* CO2 10

*Note: a timestamp will be auto-generated from above command, if user want to specify his/her own timestamp or ID. E.g: with specified time stamp: 20191012, use:*

XADD data:sensor:001 20191012 CO2 10

**Delete/expire**:

*e.g: delete/expire above entry:*

DEL/EXPIRE data:sensor:001 [time]

**Find**:

*Recommend load data to local for searching with data streaming API provided.*

**Sort**:

*Data are auto-sorted based on the timestamp*

Here is a quick view on updated permanent data schema:

