Algorithm for Distributed Architecture Keying: ADAK

Greg Beeley - LightSys Technology Services, Inc.

Overview

This document describes a scheme, ADAK, for providing coherent primary key generation in a peer-to-peer replicated environment. The scheme has the following characteristics:

1. **Compatibility.** ADAK is compatible with traditional database schemas using an autoincrement primary key, without using a composite key of any kind.
2. **Peer-to-Peer.** ADAK works peer-to-peer, where no node is required to be configured as “primary” or “master”.
3. **Graph-Connected.** ADAK works with a redundant graph-connected replication setup, where there can be cycles in the connectivity graph of the nodes.
4. **Indefinite Isolation.** ADAK allows nodes to continue to operate indefinitely, including virtually unlimited data creation and keying, while node(s) are isolated from other nodes.
5. **Avoids “Holes” in Key Sequence.** The ADAK scheme tends to avoid “holes” in the incremental key sequence, or unused key values. Such holes can occur, especially during node isolation, but the system tends to fill them in.
6. **Avoids Magic Keys.** Key values look normal, and don't contain magic number sequences. You will not need to have keys of the form NNN-XXXXXX, where NNN is the node ID and XXXXXX is the record ID created by that node. Simple integer values can be used for keys.
7. **Minimize Knowledge.** The ADAK scheme minimizes the amount of data about distant nodes that is distributed throughout the network. The hope with this principle is to limit the amount of damage caused by a compromised node or communication channel.

These characteristics make ADAK especially well-suited for Hybrid Cloud Computing systems deployed in the Majority World, where power and internet are not nearly as reliable as they are in the Western World.

Principles of Operation

The ADAK scheme operates based on keyspace partitioning. However, instead of reserving sequential ranges of key values for use at different nodes, ADAK partitions the keyspace using binary arithmetic, with keys in sequence alternating between different nodes. If only two nodes participate in the replication, and both create roughly the same number of objects, one node will create all of the even numbered keys, and the second node will create all of the odd numbered keys. To combat the inevitable “holes” that develop in such a key sequence in real life, ADAK transfers blocks of unallocated “holes” from less active nodes to more active nodes in order that any discontinuities in the key sequence can be filled in.

The ADAK scheme also depends only on peer-to-peer communication. There is no central management of the keyspace, but rather the participating nodes cooperatively ensure efficient use of the keyspace. Keyspace is managed through the passing of information peer-to-peer.

Keyspace Blocks

A keyspace of integer primary keys for a given collection (e.g. database table) is partitioned into one or more “blocks”. A block is similar in concept to a “netblock” in Internet terminology, except that the masking is inverted. On the internet, subnetting a large network into two smaller networks will result in the lower numbered half of the IP addresses going into one network, and the higher numbered half of the IP's going into the second network. With the ADAK scheme, the even numbered keys would go in one block, and the odd numbered keys would go into the second block.

Keyspace blocks are identified by the notation S/B, where S is the first available key in the block, and B is the number of suffix bits. Keyspace subblocks are identified by S-E/B, where S is the first key, E is the last key, and B is the number of suffix bits.

For illustrative purposes, we'll assume a 32-bit unsigned integer primary key. Such a key has 2^{32} different possible values. If the ADAK graph had only one node, the resulting keyspace block would consist of all possible keys, and would be notated as 0/0 (starting at 0, with 0 suffix bits). If we were to divide that block in half, the resulting blocks would be 0/1 (all even numbers) and 1/1 (all odd numbers). If we were to divide the 1/1 block in half, the resulting two blocks would be 1/2 (key sequence 1,5,9,13,...) and 3/2 (key sequence 3,7,11,15,...).

Below is a table of some example keyspace blocks:

Keyspace Block	A Part Of	Keyspace Fraction	Sequence
0/0	-	1	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,...
0/1	0/0	1/2	0,2,4,6,8,10,12,14,...
1/1	0/0	1/2	1,3,5,7,9,11,13,15,...
0/2	0/0, 0/1	1/4	0,4,8,12,...
1/2	0/0, 1/1	1/4	1,5,9,13,...
2/2	0/0, 0/1	1/4	2,6,10,14,...
3/2	0/0, 1/1	1/4	3,7,11,15,...
0/3	0/0, 0/1, 0/2	1/8	0,8,...
1/3	0/0, 1/1, 1/2	1/8	1,9,...
2/3	0/0, 0/1, 2/2	1/8	2,10,...
3/3	0/0, 1/1, 3/2	1/8	3,11,...
4/3	0/0, 0/1, 0/2	1/8	4,12,...
5/3	0/0, 1/1, 1/2	1/8	5,13,...
6/3	0/0, 0/1, 2/2	1/8	6,14,...
7/3	0/0, 1/1, 3/2	1/8	7,15,...

Keyspace Subblocks

In some cases, a keyspace block will be further subdivided by range; this is currently only used when the ADAK graph network attempts to fill in “holes”.

To create a subblock, both a starting and ending number in the range need to be specified, rather than just the starting number. For example, the range of numbers 5,6,7,8,9 would be represented as 5-9/0. The range of odd numbers between 11 and 21 would be represented as 11-21/1.

Topology: Nodes, Peers, and Channels

A “node” is a system using the ADAK scheme which creates, or may create, objects (such a database rows) in collections (such as database tables).

Two nodes become “peers” when a communication channel is set up between them. Typically this requires some kind of system administrator action, or a node connecting to a supplied or preconfigured address. Once the channel is established, messages can be transmitted by the peers over the channel.

The “ADAK graph network” is a collection of nodes and the communication channels between the nodes. It is referred to as a “graph” because it can contain cycles, i.e., redundant paths in the network.

This document does not specify how a channel should be formed or structured, but just specifies what information needs to be passed across the channels. Channels may include (list not exhaustive):

1. A socket connection, such as Web Sockets or the UNIX/BSD socket() API.

2. Messages placed in a shared filesystem folder.
3. Emails transmitted between predefined email addresses.

Key Selection

When a node needs to create a new object with an autonumbered primary key, a keyspace list for the given collection (e.g. a database table) is consulted. The node may have more than one keyspace block or subblock assigned to it for a given collection.

The rules for key selection are simple:

1. Find the lowest-numbered starting number *S* in the list of blocks/subblocks assigned to the node for the given database table.
2. Use that number as the ID for the new object (row) in the table.
3. Update the assigned keyspace block in question by incrementing *S* by 2^B, where *B* is the number of suffix bits for the keyspace block.
4. No key use information needs to be forwarded onto any other nodes (other than eventually forwarding on the newly created record in the normal course of replicated data propagation; the ADAK algorithm does not specify how to replicate data, but just specifies how to determine the key values).

For example, a node with keyspaces 10/1 and 5-7/1 that needed to create a new database row (or other object in a collection) would first look at those two keyspace blocks, and note that the second one, 5-7/1, has the lowest starting number (5 is less than 10). The number 5 is therefore the new ID for the new row (object). The second keyspace block would then be updated to 7-7/1 (by calculating, $5 + 2^1 = 7$), meaning it has only one value left in it: 7. The next create would use that 7, depleting the second keyspace block entirely, after which creates would continue at ID #10, 12, 14, and so forth, unless other keyspace blocks arrived at the node in the interim with lower starting numbers.

Obtaining Keyspace

Nodes must have keyspace in order to select key values (as described above) for new objects. There are two ways that a node can obtain keyspace:

1. Be granted one or more keyspace blocks or subblocks by the system administrator (via configuration).
2. Obtain keyspace from peer(s) through message(s) received over channel(s).

Node and Message Numbering

Each node will randomly select a 128-bit UUID as its Node ID. A high quality random data source should be used to obtain 128 bits of random data (entropy), with that data then fed into the SHA2-256 hash function. The 256-bit output of the hash function should be truncated to 128 bits and the resulting value used as the node's UUID.

Nodes should number their messages sequentially with a 64-bit message ID, starting with Message ID 1. Each channel will have its own set of message ID's, each starting with message 1.

Messages

Messages shall contain:

1. Originating (sending) Node ID
2. Destination (receiving) Node ID, set to all zeros if the peer Node ID is not yet known
3. Highest received consecutive Message ID from destination node (acknowledgment)
4. Channel state (initial startup, normal communication, or channel shutdown)
5. Timestamp

6. Message ID
7. Message type
8. Message content.

Message types include:

1. Informational messages.
2. Keyspace exchange messages.
3. Data replication messages may also be sent, but such messages are outside of the scope of this document.

Informational Messages

Informational messages communicate the status of the ADAK graph as viewed by the originating (sending) node. The receiving node uses these messages to make decisions about sharing keyspace blocks or subblocks with its peers.

1. **Informational Message:**
    1. Zero or more **Collection Information Records** which contain:
        1. Collection name (database table name, OSML pathname, etc.)
        2. Object creation rate data:
            1. Short-term keyspace allocation ratio: A_S
            2. Long-term keyspace allocation ratio: A_L
            3. Keys created per day over 24 hours (floating point value): C_S
            4. Keys created per day over 7 days (floating point value): C_L

The procedures for computing the statistics above, including keyspace allocation ratios, will be discussed later in this document.

Keyspace Exchange Message

A keyspace exchange message allows one node to pass either a subblock or block of keyspace to its peer. Since ADAK provides no way to communicate precisely with a distant non-peer node, keyspace is provided only to peers, and the peer is then responsible for either using the keyspace or passing it along further in the ADAK graph.

A keyspace exchange message consists of:

1. **Keyspace Exchange Message**
    1. One or more keyspace blocks or subblocks:
        1. Collection name
        2. Starting ID (S)
        3. Ending ID (E), if this is a subblock, zero otherwise.
        4. Suffix bits (B)

Keyspace can be lost, possibly permanently, if a keyspace exchange message is transmitted but never received by the peer, due to some failure of the channel or due to the peer ceasing to exist. To help address this issue, all messages have an acknowledgment property. If a node receives a message from a peer that indicates (via an acknowledgement value that is too small) that the peer has not received some messages, the node can re-transmit those messages after a period of time.

Keyspace Peer Initialization

So, how do multiple ADAK graph nodes end up sharing keyspace? In essence, one node is granted keyspace initially, and the keyspace is distributed throughout the graph network automatically.

Nodes will have several ADAK tunables which can be set up by the administrator, but this is not truly necessary; the ADAK scheme automatically tends to compensate for real-world usage of the keyspace.

When replication is first initiated for a collection (e.g. database table), the node will not have any keyspace to use. During the exchange of topology messages, a peer will notice that the node has no keyspace. In this situation:

1. The peer will examine its list of keyspace blocks and find the largest-sized block on the list (i.e., with the smallest bit suffix number).
2. The peer will split the block in half and provide one of the halves to the node with no keyspace. To split a block S/B in half to form two new blocks S_1/B_1 and S_2/B_2:
    1. The new prefix is one more than the original, thus $B_1 = B_2 = (B + 1)$
    2. The first block's starting ID is the same as the original: $S_1 = S$
    3. The second block's starting ID is the next in the original sequence: $S_2 = S + 2^B$

For example, the first node in the graph will be granted block 0/0 by the system administrator, via configuration. When the second node joins, the first node will split 0/0 into 0/1 and 1/1, and it will grant either 0/1 or 1/1 to the second node via a keyspace exchange message.

The nature of the ADAK scheme means that there need not be any concern about keyspace depletion at high-use nodes. As a node recognizes an imbalance between itself and a peer regarding keyspace utilization, keyspace will be exchanged between the node and its peer, to reduce the need for “holes” (keyspace subblocks) to later be exchanged. This mechanism will be covered in detail later.

Keyspace Exchange

Although the basics of ADAK are fairly simple, the challenging and complex part of the ADAK scheme is the peer-to-peer keyspace exchange based on actual object creation activity. This document proposes different ways to handle this, and these ways will need to be analyzed via simulation to decide on an implementation.

Here are some basics:

1. Keyspace is never requested explicitly. Instead, when a node notices that significantly more creation activity (relative to allocated keyspace) is happening across a peer channel somewhere (either at the peer, or at nodes beyond the peer) than is happening on the node itself, the node will grant a keyspace subblock (if the increased creation activity is only short-term) or keyspace block (if the creation activity is sustained over a longer term) to the peer in question. The peer then either uses the keyspace or passes it along.
2. To support the above, each node will need to maintain statistics about itself (regarding keyspace percentage and object creation rates) as well as obtain actionable statistics about remote activity via informational messages from peers, including information about keyspace allocation across the peer channel and object creation rates across the peer channel.
3. A significant issue is that if UUID-specific activity is not included in the informational messages, a node cannot know whether the summary activity it receives in an informational message also contains, indirectly, information about its own activity (due to a cycle in the ADAK graph). This document proposes a couple of ways to work around this limitation, but this does remain an open question.
4. The keyspace exchange algorithm needs to scale to large numbers of nodes. Although the foreseen hybrid cloud approaches at LightSys may result in networks of just a handful of nodes, we see some long-term benefit to this algorithm scaling to thousands or even millions of nodes, for use in mesh-style collaboration in the Majority World, particularly within disciple-making movements (DMMs). In order for this to work, the informational messages cannot grow significantly longer in length as the network size grows larger (i.e., they cannot contain a list of all nodes).
5. There is also value to not including the UUIDs of distant nodes in the informational messages, to help reduce the potential for this data to be used for tracking individual activity.

Informational Statistics

When a node receives an assigned keyspace block or subblock, the working assumption in ADAK is that the node is responsible for managing and using that space. No mechanism is provided in ADAK for another node to "steal" the keyspace from its owner.

However, nodes will have the ability (via processing Informational Messages) to "notice" when other nodes may be able to more effectively use keyspace, and either reactively or proactively exchange keyspace blocks or subblocks with other nodes.

To do this, nodes will track several statistics:

1. **Object Creation Rates (for self):** The node will maintain sufficient historical data to determine the number of objects it has created in the last 24 hours and in the last 7 days. This will produce C_S, the short-term object creation rate, and C_L, the long-term object creation rate. These values will be floating point values in units of keys created per day, and will be exchanged with peers via informational messages. The 24-hour and 7-day time period values are tunables for ADAK.
2. **Peer Object Creation Rates:** The node will routinely receive informational messages from peers informing the node of other nodes' object creation rates, and the node will keep the latest copy of this data for futher use.
3. **Aggregate Object Creation Rates (summary of self and all peers):** When the node transmits informational messages, it will combined its creation rate with its peers object creation rates according to a weighted average.
    1. First, the node determines the total rate for all peers.
    2. The peer total is then multiplied by the tunable ADAK network scale factor, by default 0.30.
    3. The adjusted peer total is then added to the node's own object creation rate to determine the aggregate rate (both short-term and long-term).
    4. When this aggregate rate is transmitted in an informational message, the destination peer is excluded from the peer total in (1) above. Thus, the aggregate rate shared with each peer will likely be different from that shared with other peers.
4. **Keyspace Allocation Ratio (for self):**
    1. To determine the long-term keyspace allocation ratio, A_L, only keyspace blocks (not subblocks) are considered. The ratio is determined by adding the inverse powers-of-two of all of the node's keyspace blocks. So, if one node has the 0/2 block, the ratio would be $(1/2^2)$ or 0.25. Another node with the 2/2 block and the 1/1 block would have a ratio of $(1/2^1 + 1/2^2) = (0.5 + 0.25) = 0.75$.
    2. To determine the short-term keyspace allocation ratio, A_S, both blocks and subblocks are accounted for. In this computation, we need to consider C_S, the short-term object creation rate (objects per day). $C_S + 1$ objects are allocated, in simulation (not actually changing the blocks and subblocks), from the node's keyspace blocks and subblocks, and the starting and ending key allocated in simulation are noted. The short-term ratio A_S is calculated as $A_S = C_S / (K_{S\_END} - K_{S\_START})$, where K_{S_END} is the last key allocated in simulation, and K_{S_START} is the first key allocated in simulation, and C_S is the short-term object creation rate. If C_S is zero (or rounds to zero), then a value of 1 is substituted for C_S.
5. **Aggregate Keyspace Allocation Ratios (summary of self and all peers):** These values will be computed in the same manner as the aggregate object creation rates are computed.
6. **Keyspace Demand-to-Allocation ("Provisioning") Ratio:** This final ratio, herein referred to as just the "provisioning ratio", is determined by dividing the relevant object creation rate by the relevant keyspace allocation ratio.
    1. Example: a node with a short-term creation rate C_S of 21 objects per day and a short-term keyspace allocation ratio A_S of 0.75 would have a short-term provisioning ratio P_S of $C_S / A_S = 21 / 0.75 = 28.0$.
    2. The larger this ratio is, the more object creation activity is happening relative to the node's allocated keyspace.
    3. The provisioning ratio is the most useful number for a node to use in comparing its need for keyspace in comparison to the need for keyspace across a peer channel (either at the peer, or more distantly beyond the peer). If the peer's provisioning ratio is significantly higher than the node's, then the node can consider granting some of its keyspace to the peer.

Keyspace Exchange Decisions

When a node finds that its provisioning ratio is significantly smaller than a peer's, it can make the decision to grant keyspace to the peer via a keyspace exchange.

The thresholds for such an exchange decision should be tunable ADAK parameters. A possible threshold may be when the peer's provisioning ratio is 25% higher than the node's, or perhaps 100% higher. The threshold may have to be flexible, based on the size of block that can be easily granted. For example, if a node has keyspace blocks 2/2 and 1/1, it has 75% of the keyspace. If the peer has block 0/2, or 25% of the keyspace, a useful threshold may be between 100% and 200% higher (2x to 3x as high), before the node grants the entire 2/2 block to the peer, balancing the keyspace equally. At a lesser threshold, perhaps half of the 2/2 block could be sent to the peer (i.e., either 2/3 or 6/3). Thresholds should be chosen to provide sufficient hysteresis to prevent repeated back-and-forth exchanges of keyspace.

Short-term keyspace grants of subblocks, based on the short-term ratios, can be made much more readily, and this indeed can help quickly fill in “holes” in the key sequence at a node. If the channel is reliable and has relatively high speed and low latency, it's even possible for no “holes” to ever develop at all.

Short-term grants of subblocks, however, include a complicating factor: where to set the ending value of the subblock. This may have to be determined by the object allocation rate at the peer combined with the round trip latency (and informational message transmission interval) of the channel.

Alternative(s)

The above design allows keyspace use and need to be communicated between nodes in a summary fashion, and the network scale factor of 0.30 is used to suppress the problem of cycles causing a node to try to grant keyspace, indirectly, to itself. However, that approach also will likely reduce the ability for large amounts of keyspace to be granted remotely across the ADAK graph to much more distant needs, since distant needs for keyspace will be minimized by the repeated application of the scale factor.

An alternative worthy of consideration is to, instead of (or in addition to) communicating average statistics in informational messages, to include the top two or three node-specific statistics received, based on provisioning ratios. Some kind of signature would be required on each set of statistics, so that a node could easily recognize a set of statistics as belonging to itself (and thus having traveled around the ADAK graph via a cycle in the graph). To avoid disclosing node UUIDs distantly across the graph, this could be a HMAC value based on a private random key possessed only by the node.

In this alternative, each node would pass along these top (by provisioning ratio) two or three collection information records it receives, which would also need Time To Live (TTL) values of some kind to keep the sets of statistics from being indefinitely relayed through the ADAK graph. (TTL values can either be timestamp-based or number-of-retransmissions-based, and provide a limit on how long a set of statistics can remain actively relayed through the ADAK graph)

Special Cases / Issues

There are several smallish but important issues that come up in the actual implementation of the ADAK algorithm, but many of these need to be worked out in simulation before they can be firmly decided upon.

Security

The ADAK scheme itself does not specify a specific kind of data security protection. Implementations will, without a doubt, have security requirements such as encryption and authentication. This is, however, out of the scope of this document except where it impacts the operation and design of the core algorithm.