#SASGF

VIRTUAL SAS® GLOBAL FORUM 2021

AMERICAS | MAY 18 – 20 ASIA PACIFIC | MAY 19 – 20 EMEA | MAY 25 – 26

Uniform Hashing of Arbitrary Input Into Key-Exclusive Segments

Paul Dorfman, Independent Consultant

Don Henderson, Henderson Consulting Services

Don Henderson has used SAS software since 1975, designing and developing business applications with a focus on data warehouse, business intelligence, and analytic applications. Don was one of the primary architects in the initial development and release of SAS/IntrNet software in 1996, and he was one of the original developers for the SAS/IntrNet Application Dispatcher. He is also the author of three SAS Press books including “Data Management Solutions Using SAS® Hash Table Operations: A Business Intelligence Case Study” that he co-authored with Paul Dorfman.

Problem: Input Too Large for Resources

A. **Get more resources:**

1. Request more resources (disk, memory, etc.).
2. If not enough, request more.
3. Etc.

B. **Divide-and-conquer:**

1. Segment input into a number of smaller chunks.
2. Process each segment individually.
3. Add output from each process to the final result.

Process in Segments?

- Problem: Input too large to aggregate in a *single* pass
- Can it be done in *multiple* passes?
- Need final output the same as from a single pass, e.g.:

```
select ID, Key
    , sum(Var) as SUM
    , count(distinct Var) as UCOUNT
from  Trans
group ID, Key
```

Count Distinct

- The techniques presented will focus on aggregation. However they are applicable to other data management tasks like joining and sorting data tables.

Segmented Aggregation: Need *Key-Independent* Segments

Criteria:

- **Required**: No key-value in one segment must be present in another.
- **Desired**: Nearly even number of unique key-values across all segments.

How to achieve:

- Based on a priori knowledge about the values of certain key components.
  - Such information can be obtained from the business user, or prior analysis.
  - It must be validated, which can be time consuming.
- Mapping the segments via a hash function – the focus of this presentation.

Segmentation Based on a Hash Function: Concept

Background

- Composite key-values in large inputs are diverse and numerous.
- There exists *some* combination of their bits/bytes whose values split the distinct key-values evenly according to *some* formula.
- Problem: We know *neither* the combination *nor* the formula.

Concept

- We don’t need to know!
- Instead, use a *hash function* to map the input key-values to a string *HKEY* in such a manner that:

  1. *Key-value -> HKEY* mapping is highly *random*.

  2. Each unique key-value maps *to one, and only one* unique value of *HKEY*.
- Split the unique values of some part of *HKEY* into *N* more or less equal sets.
- Use these *N* sets (e.g. in a WHERE clause) to split input into *N* segments.

Using a Hash Function

- Concatenate the key components (*via a delimiter - later on that*).
- E.g., for our sample input file *Trans*:

```
Concat = catX (‘:’,  ID,  KEY) ;
```

- Pass the result to hash function *MD5* to obtain its *signature* HKEY:

```
length HKEY $ 16 ;
HKEY = MD5 (Concat) ;
```

Or just:

```
HKEY = put (MD5 (catX (‘:’,ID,KEY), $16.) ;
```

- Function SHA256 can be used instead of MD5 - *later on that*.

Our Sample Data Trans

Obs	ID	KEY	VAR
1	B	2	1
2	B	2	2
3	B	3	2
4	A	1	3
5	A	2	1
6	A	1	3
7	B	2	3
8	B	1	3
9	A	3	2
10	B	2	2
11	B	3	1
12	A	2	3
13	B	3	2
14	A	3	2
15	A	1	3

Creating the Hash Key

- Goal: Demonstrate properties of hash signature HKEY.
- Use distinct key-values (ID,KEY) to create a test table MAP:

Could be a View
- For the Map.
- For the data to be processed.

```
proc sql ;
   create table Map as
   select distinct ID, Key
       , MD5 (catX (":", ID, Key)) as HKEY length=16 format=$hex32.
   from   Trans
   order  ID, Key ;
 quit ;
```

Hash Function Signature Properties

- Test table MAP (*hash digits of HKEY spaced for clarity*)
- Notice: HKEY byte values have a random pattern
- Can pick a byte or combination of bytes for segmentation

ID	KEY	HKEY
A	1	1A A8 1A 75 62 B7 05 FB 67 79 65 5B 8E 40 7E E3
A	2	D6 B3 D7 E5 13 1F 54 1D DE F6 81 D8 AC C1 17 13
A	3	8E 1A 7B 2F 99 09 E6 3C B6 BC D2 2E 7D E8 AB 21
B	1	0E C9 E6 87 5E 4C 6E 67 02 E1 B8 18 13 A0 B7 0D
B	2	B3 0B E9 97 C4 A0 4C 08 09 C2 5D B6 D0 A0 D3 DC
B	3	0E 04 B1 C7 15 01 16 B3 35 E8 56 60 17 29 78 63

Converting a Signature Byte into Segments

1. Pick any byte from HKEY. For example, for byte #10:

```
HBYTE = char (HKEY, 10) ;
```

2. Obtain its *rank* in [0:255] range – either expression will work:

```
RANK = rank (HBYTE) ;
RANK = input (HBYTE, pib1.) ;
```

3. Use a formula to split the ranks into segments from 1 to *N*:

```
Segment = 1 + mod (RANK, N) ;
```

Segmentation Picture for file Trans

ID	KEY	HBYTE	RANK	SEGMENT
A	1	79	121	2
A	2	F6	246	1
A	3	BC	188	3
B	1	E1	255	1
B	2	C2	194	3
B	3	E8	232	2

Segmented Aggregation: All Together

```
%macro segAgg (N=, IN=, OUT=) ;
 %let X = 1 + mod (rank (char (MD5 (catX (":",ID,Key)),10)), &N) ;
 %do SEG = 1 %to &N ;
  proc sql ;
   create table segAgg as
   select ID, Key, sum(Var) as SUM, count(distinct Var) as UCOUNT
   from   &IN (WHERE =(&X = &SEG )) group ID, Key ;
  quit ;
  proc append base=&out data=seg Agg ;
  run ;
 %end ;
%mend ;

%segAgg (N=3, IN=Trans, OUT=Agg)
```

Aggregation: Results

STRAIGHT				SEGMENTED				
ID	KEY	SUM	UCOUNT	ID	KEY	SUM	UCOUNT	Segment
A	1	9	1	A	2	4	2	1
A	2	4	2	B	1	3	1	
A	3	4	1	A	1	9	1	2
B	1	3	1	B	3	5	2	
B	2	8	3	A	3	4	1	3
B	3	5	2	B	2	8	3	

- Same data. Only *(ID,KEY) orders* are different.
- Not a problem: Aggregate files' keys are normally indexed.

More Numerous/Diverse Keys

- File Trans is too small to see the effect of MD5 on segmentation uniformity.
- Let's create a file with more numerous/diverse distinct keys (1,816 records):

```
%let N = 3 ; * Number of segments ;
%let W = 1 ; * Number of leftmost HKEY bytes ;

data ID_Key ;
  do ID = "A","B","C","D" ;
    do KEY = 1 to ceil (ranuni(1) * 1000) ;
      format HKEY $hex32. ;
      HKEY = md5 (catx (":", ID, KEY)) ;
      RANK = input (HKEY, pib&W..) ;
      Segment = 1 + mod (RANK, &N) ;
      output ;
     end ;
   end ;
run ;
```

More Numerous, Diverse Keys (*Cont'd*)

- Frequency on *Segment* with *W*=1 and *N*=(3,4):

```
proc freq data=ID_KEY noprint ;
  tables Segment / out=Segment_Freq ;
run ;
```

	Segments: N=3			Segments: N=4			
Segment	1	2	3	1	2	3	4
Count	606	610	600	451	456	452	457
Percent	33.4	33.6	33.0	24.8	25.1	24.9	25.2

Input Segmentation Works with Any Aggregation Method

- In our demo examples, SQL has been used as the aggregation method.
- Input segmentation concept applies to *any aggregation method*, such as: sort/control-break, the SAS hash object, the MEANS procedure, etc.
- Just use your method as the core of macro %segAgg. E.g., for sort/control-break just loop thru the segments as in the earlier SQL example:

```
proc sort data=&IN (WHERE=(&X = &SEG)) out=SEG ;
  by ID Key Var ;
run ;
data SEG (drop=Var) ;
  do until (last.Key) ;
    set SEG ;
    by ID Key Var ;
    SUM = sum (SUM, Var) ;
    UCOUNT = sum (UCOUNT, first.Var) ;
  end ;
run ;
```

Count Distinct

Applicability

The concept of key-independent uniform segmentation works:

- Regardless of the input data nature
- Regardless of the industry

Such as:

- Point of Sale retail data
- Financial Transactions
- Insurance Claim Data
- Social Security Payments
- So on, and so forth

Choosing the Number of Segments *N* and HKEY bytes *W*

- *N* segments reduce the demand for resources (disk, memory) ~ *N* times.
- Each extra segment means an extra pass thru input, albeit via the WHERE clause.
- Hence, *N* has to be chosen *judiciously* in order to:
  - Reduce resource usage in each pass to an acceptable level
  - Avoid overtaxing the I/O with too many passes

- Opting for a single HKEY byte (*W*=1) allows for up to *N*=256 way split.
- *W*=2 allows for up to *N*=65,536 way split.
- You are never going to need nearly as many segments (and passes).
- Practically, you may want to select:
  - *W* between 1 and 4
  - *N* as a power of 2, i.e. N=2, 4, 8, 16, etc.
  - The MOD formula will automatically handle the *N*-split regardless of *W*.

Ensuring Unique Process-Key to HKEY Mapping

- Input segmentation works because the segments are key-independent, i.e. no key-value in one segment is present in the other.
- Key-independence rests entirely on the one-to-one mapping between the process-key, such as (*ID,KEY*), and hash signature *HKEY*.
- The process-key to *HKEY* mapping includes 2 separate stages:
  - Concatenating all process-key components (let's call the result *CONCAT*).
  - Mapping of *CONCAT* to *HKEY* via a hash function.
- In order to make the mapping of process-key to *HKEY* unique:
  - The concatenation must map the process-key to *CONCAT* as one-to-one.
  - The hash function must map *CONCAT* to *HKEY* as one-to-one.
- Hence, no breach in one-to-one mapping is allowed at either stage.
- Let us consider the two stages from this standpoint, one at a time.

Concatenation Uniqueness

- Two sources of non-uniqueness:
  - CATX buffer length.
  - Improper CATX delimiting.
- CATX buffer length:
  - Is 200 by default. With long enough key-values, can result in truncation.
  - Use *LENGTH CONCAT $w* or *PUT (CATX(…),$w.)* to set the proper buffer length.
  - Choose it only as long as needed. Longer length = reduced execution speed.
- Improper CATX delimiting:
  - Never fail to use a delimiter – i.e. use CATX, not CATS.
  - Choose a delimiter *different* from the *endpoints* of any key component to avoid a delimiter-endpoint conflation.
  - Bulletproof: Surround each key component value 2 characters *different* from the delimiter. (See the paper.)

Hash Function Uniqueness

MD5:

- This hash function (16-byte signature) has a “vulnerability”: *In principle*, it can map two different arguments to the same signature (termed a *collision*).
- However, a 50% chance of getting an MD5 collision is *2**64≃2E+19*, which means:
  - To see one collision, MD5 must process 200 quintillion distinct arguments.
  - Or, it must be executed 100 trillion times per second for 100 years.
- Practically speaking, an MD5 collision is *never* going to happen.

SHA256:

- This hash function (32-byte signature) has no known collisions.
- However, it executes about *20-40 times slower* than MD5.
- Given the chance of an MD5 collision, using SHA256 for input segmentation is not worth the “peace of mind” it supposedly offers.

Thank you!

Paul Dorman, Paul.Dorfman@gmail.com

Don Henderson, Don.Henderson@hcsbi.com

Or tag one/both of us on communities.sas.com

Paul: @hashman

Don: @donh

As others can also chime in.