Paper 1748-2014

Simulation of MapReduce with the Hash of Hashes Technique

Joseph Hinson, Accenture Life Sciences, Berwyn, PA, USA

ABSTRACT

Big data is all the rage these days, with the proliferation of data-accumulating electronic gadgets and instrumentation. At the heart of big data analytics is the MapReduce programming model. As a framework for distributed computing, MapReduce uses a divide-and-conquer approach to allow large-scale parallel processing of massive data. As the name suggests, the model consists of a Map function, which first splits data into key-value pairs, and a Reduce function, which then carries out the final processing of the mapper outputs. It is not hard to see how these functions can be simulated with the SAS® hash objects technique, and in reality, implemented in the new SAS® DS2 language. This paper demonstrates how hash object programming can handle data in a MapReduce fashion and shows some potential applications in physics, chemistry, biology, and finance.

INTRODUCTION

With the rise of multimedia, social media, web traffic, digital sensors in machines, and the proliferation of mobile devices, the amount of data in today's world has been exploding beyond bounds, and analyzing such massive amounts of data has necessitated the development of novel analytic techniques and computing systems.

In 2004, Google's Jeffrey Dean and Sanjay Ghemawat introduced MapReduce, a programming model whose name was inspired by the Map and Reduce functions of functional programming, particularly the LISP programming language. With Google's MapReduce, the Map function reads a stream of data parsing it into (key, value) pairs. The Reduce function then takes the intermediate data generated by the map function and merges values associated with the same key. The model applies to large parallel data analyses where the processing can be split into smaller independent computations during the map phase and the intermediate results consolidated in the reduce phase. By breaking the processing into small tasks that can be run in parallel across a large cluster of cheap PCs running into hundreds of thousands, high performance is greatly attained.

A very popular open-source Java-based version of Map-Reduce is Hadoop, developed by the Apache Software Foundation. MapReduce programs are also written in C++, Python, Pearl, R, and Ruby.

SAS Institute also has a procedure available for SAS version 9.4 called PROC Hadoop, which accepts MapReduce codes written in Java.

The SAS hash objects, first introduced with version 9, is seeing growing applications[1-6] since its first utility in fast table look-ups and sortless merges. In this paper, the technique is used to model map and reduce algorithms. The author believes this could have real utility in the context of the DS2 language for doing parallel processing of large data.

HASH OBJECTS

SAS® currently provides three sets of DATA Step Component Objects: hash and hash iterator objects, Java objects, and logger and appender objects. These "component objects" are data elements consisting of attributes, methods, and operators. Hash objects are RAM memory-resident tables with each record in the table made up of distinct lookup key and its associated data. These objects (also called "associative arrays") permit the quick retrieval of data based on the lookup key. The name "hash" comes from the fact that each lookup key is passed through a mathematical function (called "hash") which maps the key to a record in the hash table. The hash iterator object ("hiter") is a sort of a "hash table navigator", that allows data items in a hash table to be retrieved without reliance on lookup keys. It achieves this by accessing data in a row-by-row sequential manner. Reference to Paul Dorfman's excellent comprehensive paper on hash objects is provided in this publication[2].

HASH-OF-HASHES (HOH)

Until the discovery was made by Richard DeVenezia, it was assumed hash tables could not contain other hash tables. And ever since the introduction of hash objects, the conventional practice had been to let hash tables contain just numeric and character data. But DeVenezia and Dorfman demonstrated that indeed hash tables could contain

references to other hash tables as data, using data set splitting to illustrate the concept[2]. The gem of the technique is the ability to use dynamic instantiation of hash objects (using object=_new_ hash()) to insert hash objects into other hash objects only when a new group of data is encountered. So rather than declaring a fixed number of hash objects in advance, the objects are created on the fly as needed.

Because the HOH technique is very suitable for data processing involving clustering or classification (see References 1-6), it is rather easy to see how it can model map-reduce data processing.

But being memory resident tables, memory size can become an issue in hash objects programming. However the fact that MapReduce framework involves the splitting of large data into small pieces for distributed processing makes the concept quite compatible with in-memory processing of hash objects. A single map function can be called many times to handle a small chunk of the large data.

In this paper, HOH is used to model the "mapper". The manner in which HOH is generated makes it unnecessary to have helper functions such as "shuffler", "sorter", and "combiner". The HOH process splits data into automatically-sorted key-based inner tables, whose content counts can easily be obtained with the hash attribute num_items. Even though the illustration in this paper is done with just one stream of data into HOH, one could imagine several chunks of data loading into a collection of HOHs, each yielding many sorted, classified, and clustered tables for data consolidation, as performed by a "reducer". The reduce function itself can be modeled by another HOH, which can consolidate the counts from hash tables of the same key. This paper focuses on just the algorithmic concepts underlying the MapReduce model, without regard to the hardware components of the framework.

THE DS2 LANGUAGE

This is a new programming language available in SAS 9.4 which enables the involvement of multiple processors for threading. DS2 is also unique in that instead of bringing data to the code, it rather brings the code to the data. The language comes with predefined packages such as FCMP, Hash/Hash Iterator, Matrix, and SQLSTMT. The language also allows user-defined packages.

DS2 can execute in a variety of environments: Base SAS, SAS High Performance Grid, SAS In-Memory Analytics, SAS In-Database Code Accelerator, and SAS High-Performance Analytics Server.

The Base SAS implementation of DS2 is done via PROC DS2, in which programming is done in blocks of executable code called METHOD. Programming includes both system methods and user-defined ones.

THREE SIMPLE EXAMPLES TO ILLUSTRATE SIMULATION OF MAPREDUCE BY THE HASH-OF-HASHES TECHNIQUE.

A. TEXT ANALYTICS: WORD COUNT BY CONVENTIONAL MAP-REDUCE

In whatever computer language MapReduce is written, the most popular example used for demonstration is the word count. MapReduce is used to handle the task of counting the number of occurrences of each word in a very large collection of textual documents. First, the user-coded Map function reads the document data and tokenizes the text into words. Then for each word, the mapper generates a (key, value) where the key is the word, and the value is always "1", for counting purposes (Fig. 1). The user-coded Reduce function then groups the keys together and add the values for similar keys, generating a total word count for each word. A free, unformatted text example is shown in Code 2 and Table 1, which processes the familiar prose: Desiderata. HOH automatically figures out how many inner hash tables it needs for compartmentalizing data regardless of size and content, as illustrated in Figure 2..

SIMULATION OF MAP AND REDUCE FUNCTIONS BY THE HOH PROCESS

As depicted in Figures 2 and 3, and coded in Codes 1 and 2 in the Appendix, HOH can easily mimic the Mapper and Reducer, and in a way that eliminates many intermediate steps often required in conventional MapReduce. Often, the conventional Mapper gets help from ancillary functions such as *Partitioner, Filterer, Sorter, Shuffle,* and *Combiner*.

Figure 1. Caption for Sample Figure

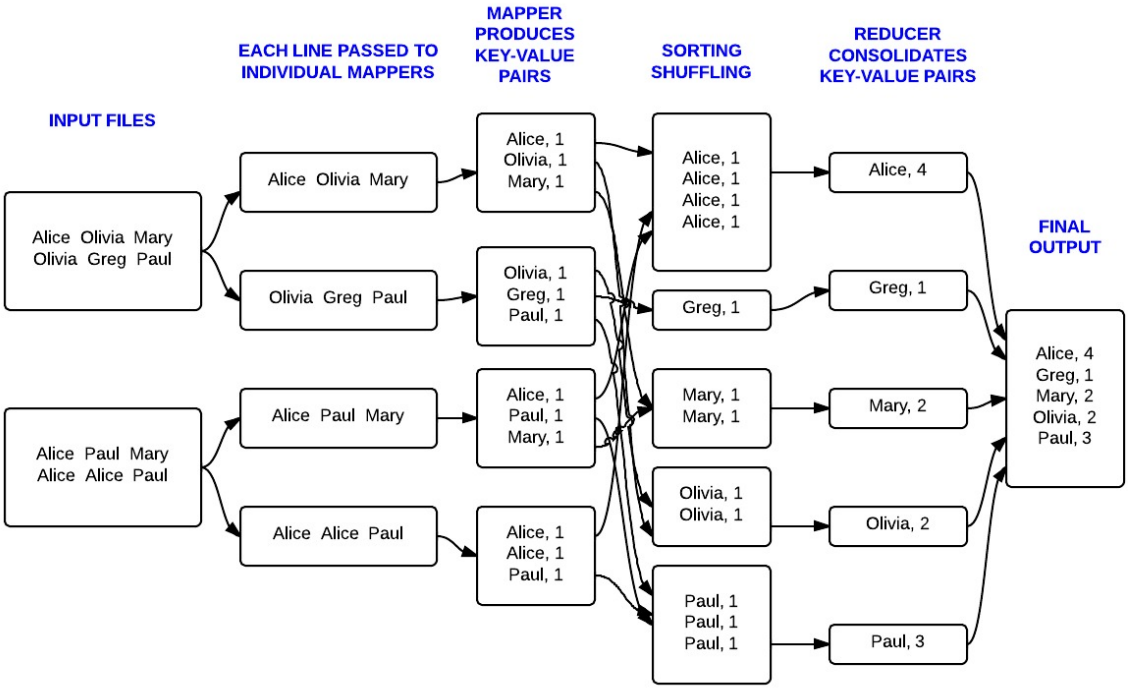


Figure 1. How Conventional MapReduce Functions Process A Word Count Task

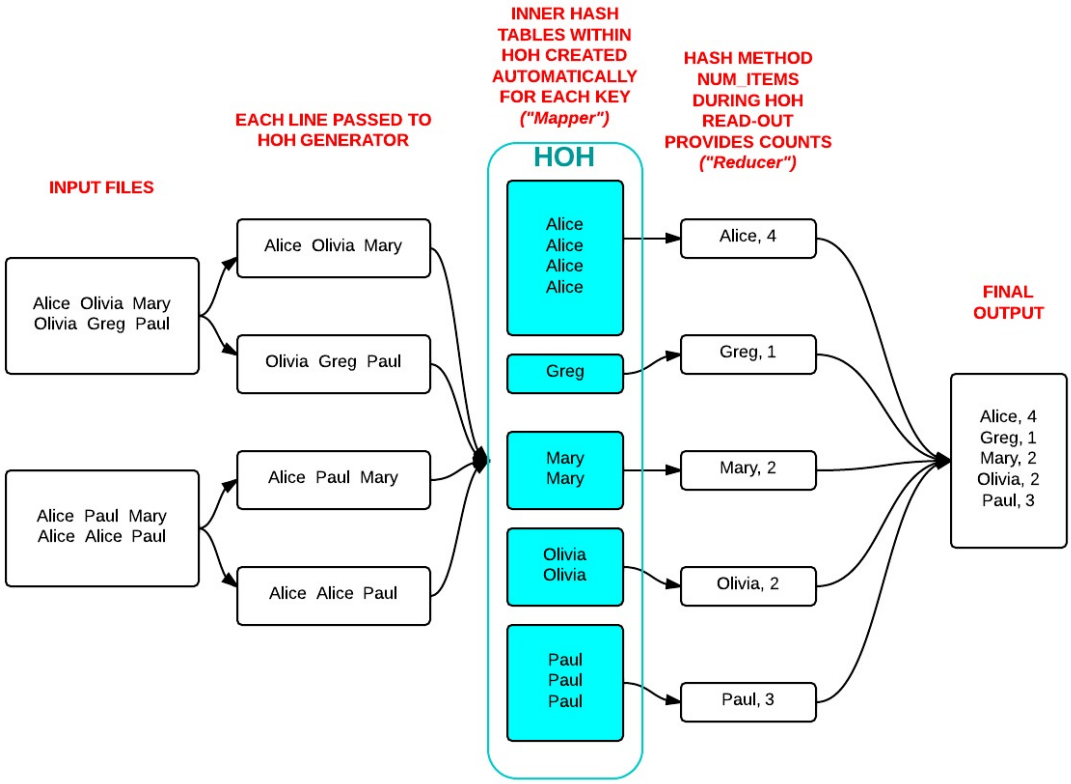


Figure 2. How The Hash-Of-Hashes Simulates The Mapper And Reducer Functions In A Word Count Task

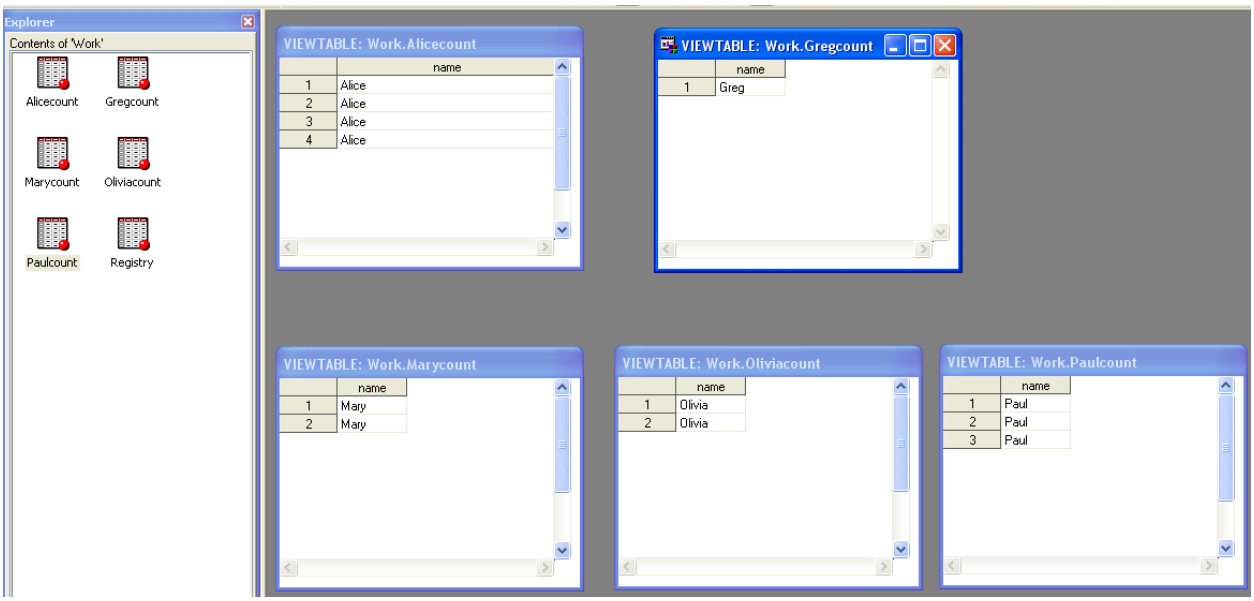


Figure 3. Inner Hash Tables Automatically Created By The Hash-Of-Hashes Process

B. WEB ANALYTICS : COUNTING UNIQUE VISITOR AD CLICK-THROUGH

With the internet explosion and ubiquity of web-enabled devices in recent times, web analytics and internet advertising assessments these days require the processing of huge multi-terabyte datasets. Such analyses become realistically feasible only with parallelization, a realization that led Google to develop in the first place, the MapReduce framework, for web page ranking and search strategies.

Let's consider a simple example involving time-stamped event data and logs of visitors' activity on a website. The data set for each visitor would have: *User-ID, IP-address, Zip Code, Ad Seen, and Ad Clicked.*

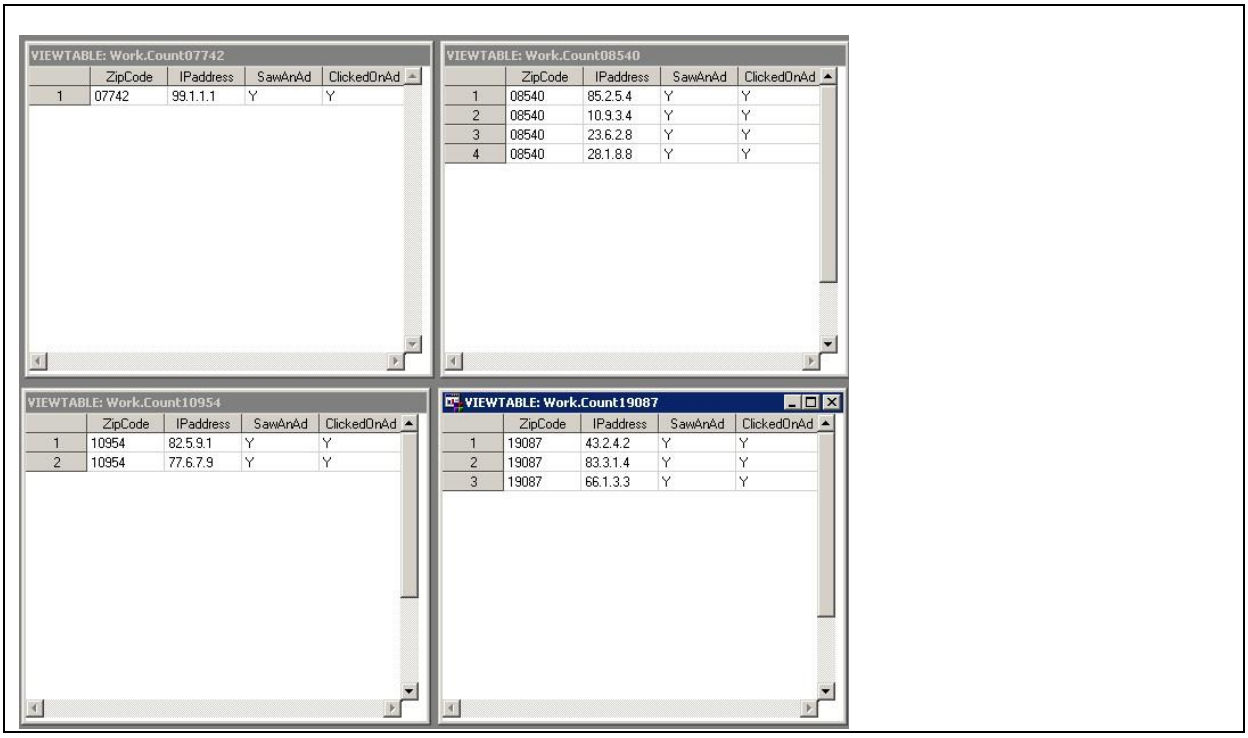
Let's suppose we need to determine how many unique visitors saw ads from each zip code and how many clicked on those ads at least once.

Since each visitors' data would be independent of other vistors, the task is suitable for parallelization, and the count processing can be distributed over a cluster of many computer nodes, each running a Map function. In this example, a Map function would read a record of data generating *key-value* pairs, where *key={zip code, IPaddress*} and *value={saw-an-ad, clicked-on-ad}*.

SIMULATION OF MAP AND REDUCE FUNCTIONS BY THE HOH PROCESS

The process of mapping and reducing can be accomplished with the hash-of-hashes (HOH) technique, as shown in Code 4 (Appendix). HOH compartmentalizes the processing, creating an inner hash table for each zip code as shown in Output 1 below:

Output 1



Output 1. Inner hash tables of unique web visitors that clicked an ad, created per zip code

Table 1:

Zip Code	Count of Unique Visitors that Clicked
07742	1
08540	4
10954	2
19087	3

Table 1. Unique Visitors That Clicked Ad Per Zip Code

C. FINANCE: FREQUENCY OF STOCK MARKET CHANGES

One common financial task is the calculation of the frequency of stock market changes. For instance, for a particular stock, one would wish to know how often in the past several years the stock changed by 1%, 2%, 3%, etc.

In the present example, the map function would need to take the opening price , the closing price, and compute the percent changes for output. Accordingly, a conventional mapper would generate *(name, value)* pairs where the name is the percent change and the value assigned '1' for counting purposes.

Simulation of such MapReduce process with HOH is demonstrated in Code 3 and its output Table 2. In this example, there was no need for the constant value of '1' for each key-value pair since HOH is easily able to provide counts through the *num_items* attribute.

THE GROWING USEFULNESS OF THE MAP-REDUCE PARADIGM

The MapReduce framework has now emerged as a highly successful programming model for large-scale (terabytes to petabytes) data-intensive computing applications. Thus its serious adoption can now be seen in many fields including science, engineering, and medicine.

1. ASTRONOMY: Image Coaddition

Advances in astronomy have been leading to very large quantities of data for analysis. In modern astronomy, a specific telescope and camera pair are used to produce a consistent database of images during large surveys of the sky. Because such surveys are conducted over long periods of time, tens of terabytes of images are generated every night. The Large Synoptic Survey Telescope (LSST) currently under construction is expected to survey 50 percent of the of the sky over for a decade, generating 30 terabytes of data per night for a total of about 60 petabytes over the 10-year operation. Such massive amounts of data would present enormous image analyses challenges without the parallel-processing approach via MapReduce. The data processing would involve extracting information such as anomaly occurrence, object classification, and object movement. This is usually accomplished via a signal averaging process called "image coaddition", which involves the alignment and stacking of multiple images of the same region in the sky, yielding a single final image with lower noise, and allowing fainter objects to be visible for study.

The MapReduce approach provides a Map function that processes a single input image (screening the image for inclusion or exclusion), with the images distributed over several computers each running a Map function. The Reduce function performs the summation of all the included bitmaps into a mosaic. Typically, a cluster of 400 computers are able to execute a Map function on 100,000 files (300 million pixels) in a few minutes!

2. HIGH ENERGY PHYSICS: Analysis of Particle Physics Experiments

Another example of data-intensive scientific computation can be found in High Energy Physics (HEP). The most well-known HEP machine is the Large Hadron Collider (LHC) with which the Higgs boson was recently discovered. Particle Physics experiments with LHC involve proton-proton collisions at very high energies. In such experiments, "events" are recorded, where an "event" contains the information from the particle collisions. The data to be analyzed consists of millions of events, with each event holding the data for a single collision and the tracks of the particles involved. Typically, an LHC experiment would generate about 40 terabytes of data per second, meaning 15 petabytes of data annually. The aim of such particle Physics experiments is to scan through all events, looking for particles with a certain mass. The final output is usually a histogram.

Rather than looping through all events serially, a more effective approach is provided by the MapReduce framework, with which events processing is conducted in a highly parallelized manner. MapReduce is applicable because each event can be processed independent of other events. With the framework, an algorithm splits the input data into individual event and passes each event to a Map function, which then evaluates the mass of the particles involved. If the mass lies in a certain range, the Map function passes on the value to the Reduce function which then would collate the results from different Map functions and generate a single output histogram.

3. BIOINFORMATICS: Sequence Alignment of Genes and Proteins

An important area of bioinformatics is sequence alignment, which involves the analysis of newly sequenced genes or protein paper, by comparing their sequence with reference genes or peptides of known sequences. This enables

biologists to predict the structure and biological function of newly-discovered genes or proteins.

The recent rapid development of “next-generation” sequencing technologies has rather caused an exponential production of sequencing data. Some late-model machines are able to generate about 50 Gigabase pairs per day of short DNA sequences called “reads”. The analysis of such huge amounts of data has become very challenging.

The popular sequencing software tool, BLAST (“Basic Local Alignment Search Tool”) created by the National Center for Biotechnology Information (NCBI) now seems somewhat inadequate for handling the huge datasets produced by the next-generation sequencers. Bioinformatics scientists have therefore now turned to the MapReduce parallelization model.

BLAST is an algorithm for comparing the nucleotides of DNA sequences or the amino-acids sequences of proteins. It compares a query sequence with a library or database of sequences, and identifies library sequences that resemble the query sequence above a certain threshold.

Running BLAST in the MapReduce framework involves:

i. Splitting the input sequence into chunks of approximately the same size
ii. Porting each chunk to a separate computer node running a Map function
iii. Making the Map function apply BLAST to compare each chunk to the gene database

4. CHEMISTRY: Identification of Peptide Sequences

Mass spectroscopy- based proteomic machines are used by chemists to identify the sequence composition of peptides, and modern high-throughput proteomic instruments generate billions of peptide mass spectra in a matter of days. To identify these experimental spectra, they are matched against a database of spectra from known protein sequences. However, as the experimental samples get diverse, the number of comparisons increases by several orders of magnitude.

Since the processing of each spectrum is independent of one another, a parallelization technique is applicable.

Using the MapReduce paradigm, the input experimental spectral data are split into roughly equal-sized chunks and supplied to each MapFunction. Typically, a cluster of about 500 computers might be involved, each running a Map function. The Map function executes an algorithm which matches the experimental sequence against a database of known peptide sequences, processing for statistical significance before passing on to the Reduce function, which then consolidates all the Map function outputs into a single output file.

5. METEOROLOGY: Global Climate Analysis

Global climate is monitored by several agencies, notably, the Goddard Institute of Space Studies (GISS), the National Oceanic and Atmospheric Administration (NOAA), and the National Climatic Data Center (NCDC). The agencies have deployed weather sensors which collect data every hour at many locations across the globe. NOAA helps maintain over 10,000 of such weather stations and provides meteorological data which include: observation date, weather-station ID, latitude, longitude, elevation, mean temperature, maximum and minimum temperature, pressure, and wind speed.

The goal of such monitoring is to keep track of the maximum recorded temperature for each year, an indication that could have a bearing on whether global warming is occurring. A typical evaluation involves data collected from 1901 to 2001.

With such enormous amounts of data being collected hourly, parallelization through MapReduce is usually adopted for data processing.

The data is split by year and distributed to a cluster of many computers, each running a Map function. For example, the function would loop through the data from 1901 to 2001, extracting the essential parameters: temperature, latitude, longitude, elevation, and mean temperature. The Map function also creates (key, value) pairs, where key=year, and value=temperature, and data output to Reduce function only if temperature is non-missing. The Map function assigns all values associated with the same key to a single Reduce function, which then computes the highest temperature for that key.

6. GEOLOGY: Oil Exploration

Sonar is typically used to locate oil deposits in the ocean. During such an exercise, a ship would drag a big sonar device in the ocean, bouncing sound waves off the ocean floor and collecting massive amounts of data about the potential geology of the ocean floor. As many vessels roam around the oceans, data accumulation becomes astronomical. However the vast majority of the data is noise.

Thus, to extract useful information, MapReduce is used for parallel processing of the sonar data for enhancement of signal-to-noise ratio.

7. MEDICINE: ECG Image Analysis

Cardiology researchers are constantly developing better techniques for detecting cardiac arrhythmias. One important approach in this effort is the rigorous analyses of the ECG waveform. In this effort, machine learning algorithms are employed for effective feature detection. However, ECG data tend to be voluminous. ECG recordings from 500 patients typically would approximate 45,000 hours of waveform data and 7 days of computational analysis time.

MapReduce is therefore employed to distribute the feature extraction analysis over a cluster of many computer processing nodes, thereby making the processing become a tiny fraction of the rime it would normally take if done serially.

CONCLUSION

The present paper has attempted to convey two main messages, that

a. the hash-of-hashes technique can be used to simulate the MapReduce programming model, as part of the parallel processing of big data, and that

b. MapReduce is now implicated in a wide array of data-intensive scientific, medical, and engineering applications.

It is quite conceivable that the HOH algorithm can be adapted for the DS2 language to enable a multiprocessing/multithreading approach to data analyses. The use of Threaded Processing in DS2 enables the processing of a huge job by splitting the job into separate tasks for parallel execution. In such a processing mode, each task is considered a thread and multiple threads can execute on one or more CPUs. With threaded processing, sections of the code can run concurrently with other sections. One threaded processing environment for DS2 ideally suited for data-intensive applications is the Massively Parallel Processing environment (MPP).

Details of the DS2 Threaded Processing can be found in Reference 7.

The question has also been raised about potential memory issues that can be encountered by using hash tables, which are memory resident. Interestingly, the future of MapReduce seems to be heading in the direction of in-memory processing[8]. It has recently become evident that physical disc access during input-output routines in the MapReduce model actually compromises speed and efficiency. Thus a hash-object approach would be quite consistent with future trends in big data parallelization techniques.

.

REFERENCES

1. Hinson, Joseph and Shi, Changhong.

   “Transposing Tables from Long to Wide: A Novel Approach Using Hash Objects“

   Proceedings of the Pharmaceutical SAS Users Group, PharmaSUG 2012, Paper PO14

2. Dorfman, Paul and Vyverman, Koen.

   “The SAS® Hash Object in Action”

   Proceedings of the SAS Global Forum 2009, Paper 153-2009

   http://support.sas.com/resources/papers/proceedings09/153-2009.pdf

3. Keintz, Mark.

   "From Stocks to Flows: Using SAS® HASH objects for FIFO, LIFO, and other FO’s"

   Proceedings of the 2011 Northeast SAS Users Group Conference, Paper

   http://www.nesug.org/Proceedings/nesug11/fi/fi03.pdf

4. Loren, Judy and DeVenezia, Richard.
"Building Provider Panels: An Application for the Hash of Hashes"
Proceedings of the SAS Global Forum 2011, Paper 255-2011
http://support.sas.com/resources/papers/proceedings11/255-2011.pdf

5. Hinson, Joseph. ”Processing of hierarchical data with hash objects Part 1 - Creation of XML documents”.
Pharmaceutical Programming, Volume 5, Numbers 1-2, December 2012, pp. 10-28(19), London, UK:
Maney Publishing.

6. Hinson, Joseph. “The Hash-of-Hashes as a ‘Russian Doll’ Structure: An Example with XML Creation”
Proceedings of the SAS Global Forum 2013, Paper 021-2013
http://support.sas.com/resources/papers/proceedings13/021-2013.pdf

7.SAS Institute. “Using Threads in DS2”.
SAS(R) 9.4 DS2 Language Reference, Second Edition
http://support.sas.com/documentation/cdl/en/ds2ref/66664/HTML/default/viewer.htm#p1m4r8j00aa5wwn1exl9bmyf8fq8.htm

8. Nadeau, Michael. ”Performing Map/Reduce In-Memory: No Hadoop Needed”.
Data Informed: Big Data and Analytics in the Enterprise, April 26, 2012.
http://data-informed.com/performing-mapreduce-in-memory-no-hadoop-needed/

RECOMMENDED READING

- Dean, Jeffrey and Ghemawat, Sanjay. “MapReduce: Simplified Data Processing on Large Clusters”.
OSDI’04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
http://research.google.com/archive/mapreduce.html

- Zhao, Jerry and Pjesivac-Grbovic, Jelena. “MapReduce: The Programming Model and Practice”.
SIGMETRICS’09 Tutorial, 2009.
http://research.google.com/pubs/pub36249.html

- MapReduce.Org
http://www.mapreduce.org/

- Google: “MapReduce In A Week”.

  Google Code University.

  http://code.google.com/edu/submissions/mapreduce/listing.html

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Joseph Hinson, PhD
Accenture Life Sciences
1160 West Swedesford Road
Berwyn, PA 19312
1-610-407-7580
joseph.w.hinson@accenture.com

APPENDIX (SAS 9.2, Windows 7)

Code 1

```
data registry;
      input name $ @@;
      datalines;
Alice Olivia Mary
Olivia Greg Paul
Alice Paul Mary
Alice Alice Paul
;
run;

*-------------------------------------------------------------------
             (1) HASH-OF-HASHES CREATION AS MAPPER
-------------------------------------------------------------------;
data _null_;
      if(1=2) then set registry;
      declare hash wd();
            wd.defineKey("name");
            wd.defineData("name","objw","hiobjw");
            wd.defineDone();
      declare hiter hiwd("wd");
      declare hash objw;
      declare hiter hiobjw;
      call missing(of _all_);

      do until(done);
            set registry end=done;
            rcf=wd.find();
            if rcf ne 0 then do;
                  objw=_new_ hash(multidata:"Y");
                  objw.defineKey("name");
                  objw.defineDone();
                  rcr1=wd.replace();
            end;
            rcr2=objw.replace();
      end;

*-----------------------------------------------------------
            (2) HASH-OF-HASHES READOUT AS REDUCER
-----------------------------------------------------------;
file "C:\Documents and
Settings\jhinson\Desktop\hashoutputs\mapreduce.rtf";
      rc=hiwd.first();
      do until (hiwd.next() ne 0);
            text=strip(left(name))||"count";
            objw.output(dataset:text);
            wcount=objw.num_items;
            put name wcount;
      end;
      stop;
run;
```

Code 1. Text Analytics: Name Count

Code 2

```
data prose;
      *** THE DESIDERATA ***;
      input word : $20. @@;
      datalines;
Go placidly amid the noise and haste and remember what peace there may
be in silence
As far as possible without surrender be on good terms with all persons
Speak your truth quietly and clearly and listen to others even the dull
and ignorant they too have their story
Avoid loud and aggressive persons they are vexations to the spirit
If you compare yourself with others you may become vain and bitter
for always there will be greater and lesser persons than yourself
Enjoy your achievements as well as your plans
Keep interested in your career however humble it is a real possession
in the changing fortunes of time
Exercise caution in your business affairs for the world is full of
trickery
But let this not blind you to what virtue there is many persons strive
for high ideals and everywhere life is full of heroism
Be yourself
Especially do not feign affection
Neither be critical about love for in the face of all aridity and
disenchantment it is as perennial as the grass
Take kindly the counsel of the years gracefully surrendering the things
of youth
Nurture strength of spirit to shield you in sudden misfortune But do
not distress yourself with imaginings
Many fears are born of fatigue and loneliness Beyond a wholesome
discipline be gentle with yourself
You are a child of the universe no less than the trees and the stars
you have a right to be here
And whether or not it is clear to you no doubt the universe is
unfolding as it should
Therefore be at peace with God whatever you conceive Him to be
and whatever your labors and aspirations in the noisy confusion of life
keep peace with your soul
With all its sham drudgery and broken dreams it is still a beautiful
world Be careful Strive to be happy
;
run;                                                                    *;
```

```
*-------------------------------------------------------------------
             (1) HASH-OF-HASHES CREATION AS MAPPER
-------------------------------------------------------------------;
data _null_;
      if(1=2) then set prose;
      declare hash wd();
            wd.defineKey("word");
            wd.defineData("word","objw","hiobjw");
            wd.defineDone();
      declare hiter hiwd("wd");
      declare hash objw;
      declare hiter hiobjw;
      call missing(of _all_);

      do until(done);
            set prose end=done;
            rcf=wd.find();
            if rcf ne 0 then do;
                  objw=_new_ hash(multidata:"Y");
                  objw.defineKey("word");
                  objw.defineDone();
                  rcr1=wd.replace();
            end;
            rcr2=objw.replace();
      end;                                                          *;
```

Code 2. Word Count from Free Text ("The Desiderata")

Table 1: Word Count (top ten) from *Desiderata*

WORD	COUNT
the	15
and	15
of	10
be	9
is	8
to	8
you	7
your	7
in	7

Code 3

```
data stockdata;
      infile datalines;
***** data from http://ichart.finance.yahoo.com/table.csv?s=BP******;
      input Date  mmddyy10. (Open High Low Close) (: 5.2) Volume
      AdjClose : 5.2;
datalines;
2010-10-15  40.92 41.11 40.40 40.62 9023100     40.62
2010-10-14  41.15 41.37 40.96 41.02 6750300     41.02
2010-10-13  41.40 41.75 41.25 41.41 6950200     41.41
2010-10-12  40.74 41.53 40.55 41.26 8343100     41.26
;
run;

*-------------------------------------------------------------------
             (1) HASH-OF-HASHES CREATION AS MAPPER
-------------------------------------------------------------------;
data _null_;
      format percentchange 5.2;
      if(1=2) then set stockdata;
      declare hash st();
            st.defineKey("percentchange");
            st.defineData("percentchange","objs","hiobjs");
            st.defineDone();
      declare hiter hist("st");
      declare hash objs;
      declare hiter hiobjs;
      call missing(of _all_);

      do until(done);
            set stockdata end=done;
            percentchange=((close-open)/open)*100;
            rcf=st.find();
            if rcf ne 0 then do;
                  objs=_new_ hash(multidata:"Y");
                  objs.defineKey("percentchange");
                  objs.defineDone();
                  rcr1=st.replace();
            end;
            rcr2=objs.replace();
      end;

*-----------------------------------------------------------
            (2) HASH-OF-HASHES READOUT AS REDUCER
-----------------------------------------------------------;
file "C:\Documents and
Settings\jhinson\Desktop\hashoutputs\mapreduce2.rtf";

      rc=hist.first();
      do until (hist.next() ne 0);
            scount=objs.num_items;
            put percentchange scount;
      end;
      stop;
run;                                                                *;
```

Code 3. Financial Stocks

Table 2: Stock Changes

% Change	Count
0.43	6
0.45	1
-1.3	6
0.35	4
1.53	7
0.23	1
0.14	5

Code 4

```
data webinfo;
      infile datalines;
      input (IPaddress ZipCode SawAnAd ClickedOnAd) ($);
      datalines;
10.9.3.4 08540 Y Y
10.5.7.1 19087 Y N
20.2.9.3 10954 Y N
23.6.2.8 08540 N N
23.6.2.8 08540 Y Y
23.6.2.8 08540 Y N
82.5.9.1 10954 Y Y
66.1.3.3 19087 Y Y
83.3.1.4 19087 Y Y
43.2.4.2 19087 Y Y
77.6.7.9 10954 Y Y
28.1.8.8 08540 Y Y
23.6.2.8 08540 Y N
82.5.9.1 10954 Y Y
66.1.3.3 19087 Y Y
83.3.1.4 19087 N N
43.2.4.2 19087 Y N
10.5.7.1 19087 Y N
20.2.9.2 10954 N N
23.6.2.8 08540 Y N
85.2.5.4 08540 Y Y
66.2.6.8 08540 Y N
99.1.1.1 07742 Y Y
;
run;
*;
```

```
data _null_;
      length wbcount 8;
      if(1=2)then set webinfo;
      declare hash wb (ordered:"Y");
      rc=wb.defineKey("zipcode");
      rc=wb.defineData("zipcode","objip");
      rc=wb.defineDone();
      declare hiter hiwb("wb");

      declare hash ct (ordered:"Y");
      rc=ct.defineKey("zipcode");
      rc=ct.defineData("zipcode","wbcount");
      rc=ct.defineDone();

      call missing(of _all_);

      declare hash objip;

      do until(done);
            set webinfo end=done;
            if (clickedonad="Y")then do;
                  rcf=wb.find();
                  if (rcf ne 0)then do;
                        objip=_new_ hash(multidata:"Y");
                        objip.defineKey("ipaddress");

      objip.defineData("zipcode","ipaddress","sawanad","clickedonad");
                        objip.defineDone();
                        rp1=wb.replace();
                  end;
                  rp2=objip.replace();
            end;
      end;


**********HOH readout**********;
      length text $12;
      rch=hiwb.first();
      do until (hiwb.next() ne 0);
            text="count"||strip(left(zipcode));
            wbcount=objip.num_items;
            ct.ref();
            rco=objip.output(dataset:text);
            call missing(text,wbcount,zipcode,ipaddress);
      end;
      rcc=ct.output(dataset:"summary");
      stop;
run;
```

Code 4. Computation by Hash-of-Hashes of Unique Visitors Per Zip Code That Clicked on Ad

Table 3:

Zip Code	Count of Unique IP Address
07742	1
08540	4
10954	2
19087	3

Table 3. Web Analytics Summary: Unique Visitors Per Zip Code That Clicked Ad