CEO: A Cloud Epistasis cOmputing model in GWAS

Zhengkui Wang, Yue Wang
NUS Graduate School for Integrative Sciences and Engineering
National University of Singapore, Singapore
{*wangzhengkui,wangyue*}*@nus.edu.sg*

Kian-Lee Tan, Limsoon Wong
School of Computing
National University of Singapore, Singapore
{*tankl,wongls*}*@comp.nus.edu.sg*

Divyakant Agrawal
Department of Computer Science
University of California, Santa Barbara, USA
agrawal@cs.ucsb.edu

[1]

***Abstract*—The 1000 Genome project has made available a large number of single nucleotide polymorphisms (SNPs) for genome-wide association studies (GWAS). However, the large number of SNPs has also rendered the discovery of epistatic interactions of SNPs computationally expensive. Parallelizing the computation offers a promising solution. In this paper, we propose a cloud-based epistasis computing (CEO) model that examines all k-locus SNPs combinations to find statistically significant epistatic interactions efficiently. Our CEO model uses the MapReduce framework which can be executed both on user's own clusters or on a cloud environment. Our cloud-based solution offers elastic computing resources to users, and more importantly, makes our approach affordable and available to all end-users. We evaluate our CEO model on a cluster of more than 40 nodes. Our experiment results show that our CEO model is computationally flexible, scalable and practical.**

***Keywords*-GWAS; Cloud computing; MapReduce; Hadoop**

I. Introduction

Discovering epistatic interactions of single nucleotide polymorphisms (SNPs) is becoming increasingly important and challenging. From the scientific viewpoint, such interactions capture the effects of multiple genetic variations that result in complex diseases (e.g, breast cancer, diabetes and heart attacks). Thus, finding these interactions is a first step towards understanding the cause of these diseases.

From a computational perspective, it incurs prohibitively high computation overhead to determine the interactions of SNPs. Given n SNPs, the number of k-locus is ${}^nC_k = \frac{n!}{k!(n-k)!}$. Now, typical values for n is hundreds of thousands. In fact, the dataset from the Hapmap project [1] contains 3.1 million SNPs, and the 1000 Genome project [2] provides about 9 million new SNPs. These render existing statistical modeling techniques (which work well for small number of SNPs) [3] [4] [5] [6] impractical. Likewise, techniques that enumerate all possible interactions [7] [8] are not scalable for large number of SNPs. To reduce the computation overhead, heuristics [3] [4] [5] have also been developed. These schemes add a filtering step to select a fixed number of candidate epistatic interactions and fit them to a statistical model. However, these approaches risk missing potentially significant epistatic interactions, because the selection criterion in the filtering step may be biased towards certain epistatic interaction.

A promising solution to the computation challenge is to exploit parallel processing. This is the approach we adopt in this paper. [9] provided a tool for processing only single-locus and two-locus SNPs analysis using supercomputer system on selected types of processors and compilers. However, it is not easy for researchers to rewrite their own programs on specialized hardware. Instead, we aim to develop tools for k-locus analysis with high scalability which can be easily deployed on any affordable PC-based cluster. There are already low-cost commercially available cloud platform (e.g., Amazon EC2) where our techniques can be deployed and made accessible to all. The pay-as-you-use model of such commercial platforms also makes them attractive for end-users.

In this paper, we propose a cloud-based epistasis computing (CEO) model to find statistically significant epistatic interactions. Our solution is based on Google's MapReduce framework [10], and implemented over Hadoop [11], an open source equivalent implementation of the MapReduce framework. We develop solutions for determining significant interactions for two-locus and three-locus as well as computing the top-k most significant answers efficiently. As a first cut, we have adopted a brute force approach that examines all possible interactions among the SNPs. This ensures that we will not miss any statistically significant interactions. Our method can be easily extended to deal with heuristics approaches. We validate our proposed CEO model on a local cluster of more than 40 nodes. Our results show that our CEO model is efficient, and that the MapReduce framework can be effectively deployed for bioinformatics research such as the GWAS.

The rest of the paper is organized as follows. Section II provides the problem formulation and reviews some back-

[1]This paper is published in IEEE International Conference on Bioinformatics & Biomedicine,PP. 85-90, Dec 2010, HongKong

ground knowledge. In Section III, we propose our CEO processing model for both two-locus and three-locus analysis. Section IV reports results of a performance study on our own cluster. In Section V, we also present an efficient approach to retrieve the top-k most significant answers, and finally, we conclude this paper in Section VI.

II. Background

In this section, we give the problem formulation, followed by the MapReduce architecture and programming model.

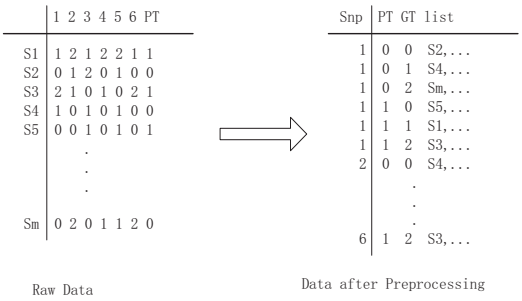


Figure 1. Data Formats before and after Preprocessing

A. Problem Formulation

Typically a GWAS uses two types of data - genotype data that codes the genetic information of each individual, and phenotype data that measures the individual's quantitative traits. For simplicity, we use the genotype data which is bi-allelic (i.e., a locus has allele A and T which can form three types of genotypes, AA, AT and TT.) and is encoded as 0, 1 and 2 in the raw data. For phenotype data, we consider the binary form (0 for control and 1 for case). Our model can handle other types of genotype and phenotype data also. The figure on the left of Figure 1 shows an example of the raw data format for a dataset with m samples and 6 SNPs. Each row contains the individual sample information of raw data. The first and last columns are the sample id and phenotype. The rest of the columns are the genotype of each SNP.

For our scheme to work, the raw data has to be pre-processed to transform the SNP information into the following new data format: $< SNP_i, PT, GT, list(sampleID) >$ where SNP_i, PT and GT are the i^{th} SNP, phenotype value and the SNP genotype respectively. $list(sampleID)$ stores all the sample ids in the data set whose phenotype and SNP genotype on the SNP_i are PT and GT respectively. The figure on the right of Figure 1 depicts the transformed data.

The pre-processing can be performed in one MapReduce job efficiently. For example, pre-processing 100,000 SNPs only takes 76 seconds on a 43-node cluster. As this is not the focus of our work, we shall not discuss this further. For the rest of this paper, we assume that the input to our algorithm is the pre-processed data.

The goal of our research is to identify a set of most significant SNP pairs (epistatic interactions) that correlate to the phenotype. To measure the association between different orders epistatic interaction and phenotype in our CEO model, we adopt the χ^2-test [12], which is widely used. Moreover, as our CEO framework assumes no statistical model fitting and thus parameter free, the χ^2-test is effective in capturing interactions of arbitrary order.

Take two order epistatic interactions as an example. Let $n_{0(j,k)}$ denote the number of samples in the control group whose first locus's genotype code is 'j' and second locus's genotype code is 'k', where j and k take on values 0, 1 or 2. Likewise, we can denote $n_{1(j,k)}$ for the case group. For two-locus, we have 18 combinations ($2 \times 3 \times 3$). Moreover, let $n = \sum_{i=0}^{1}\sum_{j=0}^{2}\sum_{k=0}^{2} n_{i(j,k)}$, $n_i = \sum_{j=0}^{2}\sum_{k=0}^{2} n_{i(j,k)}$, and $n_{j,k} = \sum_{i=0}^{1} n_{i(j,k)}$. The null hypothesis behind the χ^2-test is that there is no association between two-locus epistatic interaction and phenotype. We can calculate the χ^2-test value of this epistatic interaction using the following formula:

$$\chi^2 = \sum_{i=0}^{1}\sum_{j=0}^{2}\sum_{k=0}^{2} \frac{(n_{i(j,k)} - n_i n_{(j,k)}/n)^2}{n_i n_{(j,k)}/n}$$

As the χ^2-test statistic follows the χ^2 distribution, thus corresponding significance level can be obtained after Bonferroni correction. The lower the value is, the more confident we are to reject the null hypothesis.

The resultant p-value for the two-locus epistatic interation can be obtained as $P(x>C)$ where C is the χ^2-test value, and P(x) is the probability at value x under the χ^2 distribution.

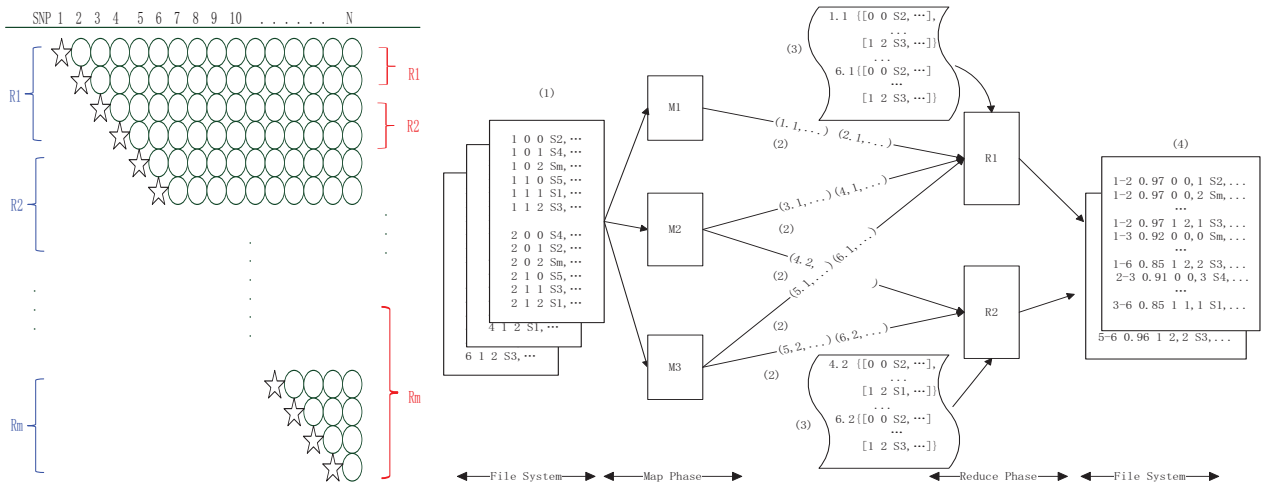
The above expressions can be easily generalized for three-locus interaction. We shall omit that due to space constraints.

B. The MapReduce Architecture and Programming Model

Under the MapReduce framework, the system architecture of a cluster consists of two kinds of nodes, namely the NameNode and DataNode. The Namenode works as a master of the file system, and is responsible for splitting data into blocks and distributing the blocks to the data nodes (DataNodes) with replication for fault tolerance. A JobTracker running on the NameNode keeps track of the job information, job execution and fault tolerance of jobs executing in the cluster. A job may be split into multiple tasks, each of which is assigned to be processed at a DataNode.

The DataNode is responsible for storing the data blocks assigned by the NameNode. A TaskTracker running on the DataNode is responsible for the task execution and communicating with the JobTracker.

The MapReduce computational paradigm divides the processing job into small tasks, each of which runs on different nodes to parallelize the processing in a large cluster. The computation of MapReduce follows a fixed model with a `map` phase followed by the `reduce` phase. The MapReduce library is responsible for splitting the data into chunks and distributing each chunk to the processing units (called



(a)SNP-pairs representation and distribution to reducers

(b)Two-locus epistatic analysis example with 6 SNPs

Figure 2. Examples of data preprocessing and two-locus epistatic analysis using CEO model

`mappers`) on different nodes. The `mappers` process the data read from the file system and produce a set of intermediate result which will be shuffled to the other processing units (called `reducers`) for further processing. Users can set their own computation logic by writing the `map` and `reduce` functions in their applications.

Map phase : The `map` function is used to process the $(key, value)$ pairs $(k1, v1)$ which are read from data chunks. Through the `map` function, the input set of $(k1, v1)$ pairs are transformed into new set of intermediate $(k2, v2)$ pairs. The MapReduce library will sort and partition all the intermediate pairs and pass them to the `reducers`.

Shuffling phase : The partitioning function is used to partition the emitted pairs from the `map` phase into M partitions on the local disks, where M is the total number of `reducers`. The partitions are then shuffled to the corresponding `reducers` by the MapReduce library. Users can specify their own partitioning function or use the default one.

Reduce phase : The intermediate $(k2, v2)$ pairs with the same key that are shuffled from different `mappers` are sorted and merged together to form a values list. The key and the values list are fed to the user-written `reduce` function iteratively. The `reduce` function makes a further computation to the key and values and produces new $(k3, v3)$ pairs. The output $(k3, v3)$ pairs are written back to the file system.

III. CEO Processing Model

In this section, we introduce our CEO processing model using MapReduce. In our system, we provide several components including two-locus epistatic analysis and three-locus epistatic analysis.

A. Two-locus epistatic Analysis

For two-locus epistatic analysis, we aim at finding statistically significant interaction among all SNP pairs. For each pair of SNP combination, the p-value is computed (as described in Section II) to determine its signficance. For N SNPs in the data set, we need to calculate $\frac{N(N-1)}{2}$ two-locus SNPs combinations, as depicted in Figure 2(a). Each row represents a subset of SNP-pair computations where the starred node has to be paired up with a circled node. Thus, row 1 has (N-1) pairs, row 2 has (N-2) pairs and so on.

Our goal essentially is to split these $\frac{N(N-1)}{2}$ pairs of SNPs across all nodes to be processed in parallel. We have two issues to address here: (a) How do we split the SNP pairs across all nodes? (b) How to perform two-locus analysis under the MapReduce framework?

We shall first look at issue (a). Given N SNPs and M `reducers`, we consider the following two simple strategies:

Naive Model. The most straightforward approach is to simply distribute approximately equal number of rows to each `reducer`. This is depicted by the square brackets on the LHS of Figure 2(a) where the first $\frac{N}{M}$ rows are assigned to the first `reducer`, the next $\frac{N}{M}$ rows are assigned to the second `reducer` and so on. Here, the number of SNP-pairs can be easily determined without any additional meta-data, e.g., for row 1, we know that we need to pair up SNP_1 (starred node) with all other remaining SNPs (circled node), resulting in (N-1) pairs.

Greedy Model. Under the naive model, some `reducers` are more heavily loaded than others, e.g., `reducer` one is likely to be a bottleneck. To achieve better load balancing, we also examine a greedy solution. Ideally, each `reducer` should process $\frac{N(N-1)}{2M}$ SNP pairs. Therefore, starting from the first row, we seek to allocate consecutive rows to a

reducer such that the total number of SNP pairs for these rows is closest to $\frac{N(N-1)}{2M}$. In Figure 2(a), the square brackets on the RHS show that, under the greedy scheme, each reducer may be assigned different number of rows to process. However, the computation task in each reducer is about the same. Like the naive scheme, this method also requires minimum meta-data to be transferred.

As we shall see in our experimental study, (see Section IV), it turns out that these schemes are surprisingly effective (in the sense that the processing cost is almost proportional to the number of pairs/triples to be computed).

We are now ready to look at issue (b). WLOG, let us assume we have M reducers. Under the MapReduce framework, the mapper essentially determines the reducer in which a SNP pair should be sent to, and the reducer computes the statistical significance of each SNP pair allocated. Figure 2 (b) shows how the CEO model processes the data having 6 SNPs.

Map Phase: Each mapper reads a chunk of the input (pre-processed) data. For each SNP, it then determines the reducers which this SNP should be shuffled to. We shall discuss how the reducers are determined later. It suffices now to assume that this information is available to the mapper. We note that one SNP information may be shuffled to multiple different reducers. For example, in Figure 2(a), SNP_N needs to be shuffled to all the reducers. This, unfortunately, is not supported by the MapReduce framework which allows only one output $(key, value)$ pair emitted from a mapper to be shuffled to one reducer.

We resolve this problem by replicating and emitting as many copies of a SNP as required. In addition, each such pair is "tagged" with the corresponding reducer identifier to distinguish the reducer that the pair should be shuffled to. In other words, for each reducer for which an SNP, SNP_i, should be shuffled to, we generate and emit a $(key, value)$ pair where key is set as $SNP_i.reducer\_marker$ ($reducer\_marker$ is the identifier of the reducer that this SNP_i should be shuffled to), as shown in the Figure 2(b) subgraph (2), and value contains the rest of the SNP information including the genotype, phenotype and the sample id list. In this way, all the output $(key, value)$ pairs with the same $reducer\_marker$ are shuffled to the same reducer.

Shuffling Phase: We write our own partitioning function to parse the $reducer\_marker$ in the key and partition the emitted pairs to multiple reducers.

Reduce Phase: The MapReduce library sorts and merges the intermediate result based on the key. The $(key, value)$ pairs with the same key, are grouped together as $(key, set(values))$ pair where $set(values)$ is a set of values for that key, as shown in Figure 2(b) subgraph (3). The $(key, set(values))$ pairs are supplied to user's reduce function in sorted order. Because all the keys at the reducer have the same reducer_marker, the keys will be sorted only based on the SNP_i. Thus, the data for SNP_i are sent to the reduce function before those for SNP_j where $i < j$. This means that the starred nodes are supplied earlier than the circled nodes. Therefore, in each reducer, only the starred nodes need to be cached in the main memory. As the circled nodes are received, they can be immediately paired up with the starred nodes to compute its p-value, after which the circled nodes can be discarded. As such, our CEO model significantly reduces the memory utilization.

In our processing model, the two-locus analysis finishes in one MapReduce job.

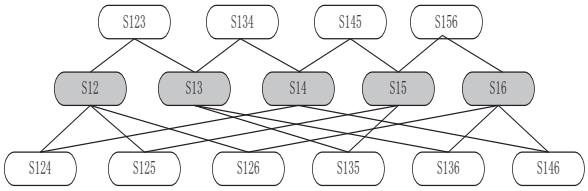


Figure 3. All the Three-locus SNPs having SNP1

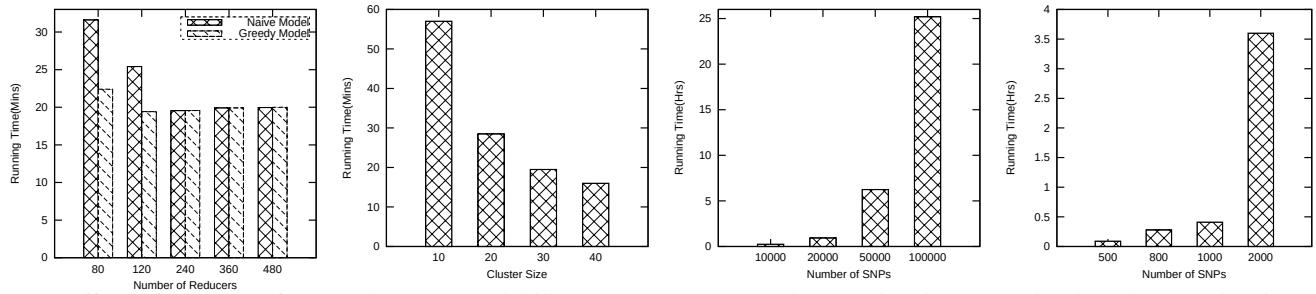
B. Three-locus epistatic Analysis

Three-locus epistatic analysis aims at finding statistically significant interaction between three SNPs. Here we propose one way of doing three-locus epistatic analysis using the output of two-locus epistatic analysis. Note that the output data of two-locus analysis are written to the file system.

As what we have discussed before, from each row in Figure 2(a), we can get all the needed two-locus SNPs combinations involving the starred node SNP. Further, if we combine any two two-locus SNPs from one row, we can get all possible three-locus SNPs involving the starred node SNP. Figure 3 shows an example of finding all the three-locus SNPs with SNP_1 using the two-locus SNPs information from the first row in 6 SNPs example. In the same way, all the possible three-locus SNPs involving SNP_m can be generated from the combinations of two-locus SNPs of the row whose starred node is SNP_m. For three-locus epistatic analysis, the same processing model can be adopted here. All the two-locus SNPs information which are derived from the same row need to be processed in the same reducer to get all the three-locus SNPs.

Map phase: The two-locus SNPs data is split into small chunks and each chunk is assigned to each mapper by the MapReduce library. As what has been mentioned above, the two-locus SNPs derived from the same row must be shuffled to the same reducer. To achieve this, the key in the output $(key, value)$ pair from Map phase is set as $SNP_i.SNP_j$, where SNP_i and SNP_j are the starred node and the circled node respectively.

The advantage of setting the output key in this format is that, after sorting the intermediate result according to



(a) Effect of number of reducers on completion time

(b) CEO Scalability on different clusters

(c) Two-locus epistatic analysis on 43-node cluster

(d) Three-locus epistatic analysis on 43-node cluster

Figure 4. (a), (b) and (c) are the evaluation results for two-locus epistatic analysis. (d) is the performance evaluation result for three-locus epistatic analysis

the keys by MapReduce library, all the two-locus SNPs from one row can be grouped closely and fed into the `reduce` function continuously. In the `reduce` phase, after processing all the two-locus SNPs from one row, the data can be discarded from the memory to minimize the memory utilization.

Shuffling phase: Our specified partitioning function is used to partition the pairs according to SNP_i value in the integer part of the key. The intermediate result from the `mappers` with the same SNP_i will be shuffled to the same `reducer`.

Reduce phase: After sorting and merging the intermediate result, the two-locus SNPs information with smaller starred node, will be supplied to `reduce` function earlier than the others. Combining any two two-locus SNPs at the `reducer`, we get the three-locus SNPs and calculate its statistical significance. The result is then output to the file system.

The load balancing algorithm can also be used here for optimization. Three-locus analysis can be performed using one MapReduce job using the two-locus SNPs data.

IV. Experiments and Results

Apache Hadoop is an open source equivalent implementation of the MapReduce framework, running on HDFS (Hadoop distributed file system). We run a series of experiments on our local cluster of more than 40 nodes to evaluate our model in Hadoop. Each node consists of aX3430 4(4) @ 2.4GHZ CPU running Centos 5.4 with 8GB memory and 2x 500G SATA disks. Moreover, since our tasks at hand are computationally intensive, we set the number of `reducers` per node to be equal to the number of cores at the node, which is 4. This guarantees that each `reducer` can get one core. Therefore, there are a total of 4*N `reducers` which can be run simultaneously on a N-node cluster.

Effect of number of reducers: For Hadoop application, a user can specify the number of `reducers` to be used in one job. Because we have preconfigured the total number of `reducers` to be 4*N for a N-node cluster, this may require multiple phases to complete a job. For example, if N=30, then by specifying 120 `reducers` in one job, we can complete it in 1 phase; with 360 `reducers`, it will then take 3 phases to complete the job. Our first experiment is to investigate the optimal number of `reducers` that should be set for one job based on a given cluster size. This experiment is conducted with a 10,000 SNPs dataset on a 30-node cluster. Note that all the datasets we used include 2000 samples. Figure 4(a) presents the running time for both the Naive and Greedy models. As shown, there is a certain optimal number of `reducers` that should be used. When the number of `reducers` is too small, the computation resources are not fully utilized. On the other hand, when the number of `reducers` is too large, the processing may require multiple phases which increases the communication overhead. We note that while the Greedy model is optimal when the number of `reducers` corresponds to the actual configured value (i.e., 120), the Naive model is optimal when a larger number of `reducers` is used (i.e., 240). This is because for the Naive model, a larger number of `reducers` means that the `reducer` with the most skewed load will be allocated smaller load. In fact, as the number of `reducers` increases, the Naive scheme performs as well as the Greedy model.

Based on the above results, for the following experiments, we only use the Greedy model.

Scalability: In this experiment, we study the scalability of the CEO model as the system resources increase. Figure 4(b) shows the completion time analyzing 10,000 SNPs as the cluster sizes increases from 10 to 40 nodes. The `reducer` numbers in each job are set as 40, 80, 120 and 160 respectively. From the result, we can see that when more nodes are added for processing, the completion time reduces. In fact, we observe a linear speedup in performance. When we double the resources, the execution time reduces to half, such as the execution time on 10/20/40-node clusters.

Two-locus analysis: In this experiment, we study the performance of the CEO model for two-locus analysis as we vary the number of SNPs processed. Figure 4(c) shows the processing time for the data sets with 10,000, 20,000, 50,000 and 100,000 SNPs on a 43-node cluster. As expected, the

processing time is essentially proportional to the number of interacting SNP-pairs to be evaluated. We observe that even for 100,000 SNPs, the CEO model only takes 25 hours to complete the processing. This shows that our CEO model is effective.

Three-locus analysis: We also evaluated the performance of three-locus analysis on the 43-node cluster. The result is presented in Figure 4(d) for SNP size of 500, 800, 1,000 and 2,000. We observe that the running time is also proportional to the number of SNP-triples. This confirms that the CEO scheme can effectively balance the load across all nodes.

V. Top K Retrieval

In our system, we store the result of the two-locus and three-locus analysis in HDFS to allow users to do further analysis. One important function that we can further provide is to allow users to retrieve only the top-k most significant results with the lowest p-value. We have also provided such a capability in our system under the MapReduce framework. The basic idea is to split the output of the two/three-locus analysis into chunks. Each chunk is then assigned to one `mapper`. Next, each `mapper` will select the top-k most significant pairs/triples and shuffled these results to one `reducer`. Finally, the `reducer` can determine the global top-k answers based on all local top-k ones it receives. Our top-k scheme is very efficient. For example, retrieving the top 10 most significant SNPs information from the two-locus output in the size of 54GB only takes 132 seconds in the 43-node cluster.

VI. Conclusion

According to [9], it would require 1.2 years to do the pairwise epistasis testing of 500,000 SNPs using the serial program on a 2.66 GHz single processor without parallel processing. In this paper, we have provided a cloud epistatic computing model (CEO) for large scale epistatic interactions using the MapReduce framework. Our experimental results demonstrated the practical advantage of using the CEO model to exhaustively search two-locus epistatic interaction. We also provided a three-locus analysis approach as an example of k-locus analysis using our model.

More importantly, by using the MapReduce framework, we have shown that large scale data analysis in GWAS can be easily performed over commodity computers or cloud resources. The scalability of the MapReduce framework to thousands of machines with good fault tolerance will make such compute-intensive computations acceptable.

Currently, we have used the popular χ^2-test to measure the interaction effect. Our CEO model can be easily adapted to handle other methods that utilize contingency table information to obtain interaction effect (eg,. likelihood ratio, normalized mutual information, uncertainty coefficient). Also, for the existing methods involving a filtering step and statistical model fitting step, our work can be used as a filtering step to retrieve the top-k most significant interactions for follow-up analysis.

As future work, we plan to look at other strategies to allocate SNPs to nodes. For example, a *Best-fit Model* may assign the next available row to the `reducer` with the least number of SNP pairs. Alternatively, an *Ideal Model* may assign the SNP pairs in a round robin fashion to the `reducers` so that every `reducer` will end up with the same number of SNP-pairs. We will explore such methods to study their effectiveness.

The CEO source code can be downloaded at $http : //www.comp.nus.edu.sg/ \sim wangzk$.

Acknowledgment

Zhengkui Wang and Yue Wang are supported by the NUS NGS scholarships.

References

[1] Hapmap Project: http://hapmap.ncbi.nlm.nih.gov/

[2] 1000 genome Project: http://www.1000genomes.org/page.php

[3] T. T. Wu, Y. F. Chen, T. Hastie, E. Sobel, and K. Lange. Genome-wide association analysis by lasso penalized logistic regression. *Bioinformatics*, 25(6):714–721, mar 2009.

[4] M. Y. Park and T. Hastie. Penalized logistic regression for detecting gene interactions. *Biostat*, 9(1):30–50, jan 2008.

[5] J. Wu, B. Devlin, S. Ringquist, M. Trucco, and K. Roeder. Screen and clean: a tool for identifying interactions in genome-wide association studies. *Genetic Epidemiology*, 34(3):275–285, apr 2010. PMID: 20088021.

[6] C. Yang, X. Wan, Q. Yang, H. Xue, and W. Yu. Identifying main effects and epistatic interactions from large-scale SNP data via adaptive group lasso. *BMC Bioinformatics*, 11(Suppl 1):S18, 2010.

[7] X. Zhang, S. Huang, F. Zou, and W. Wang. TEAM: efficient two-locus epistasis tests in human genome-wide association study. *Bioinformatics*, 26(12):i217–227, Jun 2010.

[8] X. Wan, C. Yang, Q. Yang, H. Xue, X. Fan, N. L. S. Tang, and W. Yu. BOOST: a fast approach to detecting gene-gene interactions in genome-wide case-control studies. *1001.5130*, Jan 2010.

[9] L. Ma, H. B. Runesha, D. Dvorkin, J. Garbe, and Y. Da. Parallel and serial computing tools for testing single-locus and epistatic SNP effects of quantitative traits in genome-wide association studies. *BMC Bioinformatics*, 9(1):315, 2008.

[10] J. Dean, S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Proceedings of the 6th symposium on operating systems design and implementation (OSDI)*, 137-150, December 2004

[11] Apache Hadoop Project: http://hadoop.apache.org/

[12] D.J. Balding. A tutorial on statistical methods for population association studies. *Nature Reviews Genetics 7*,781-791, October 2006