BIOINFORMATICS ORIGINAL PAPER

Vol. 27 no. 8 2011, pages 1045–1051
doi:10.1093/bioinformatics/btr091

Genome analysis

Advance Access publication March 2, 2011

eCEO: an efficient Cloud Epistasis cOmputing model in genome-wide association study

Zhengkui Wang[1,*], Yue Wang[1], Kian-Lee Tan[1,2], Limsoon Wong[1,2] and Divyakant Agrawal[3]

[1]NUS Graduate School for Integrative Sciences and Engineering, [2]Department of Computer Science, School of Computing, National University of Singapore, Singapore and [3]Department of Computer Science, University of California, Santa Barbara, 93106-5110, USA

Associate Editor: Martin Bishop

ABSTRACT

Motivation: Recent studies suggested that a combination of multiple single nucleotide polymorphisms (SNPs) could have more significant associations with a specific phenotype. However, to discover epistasis, the epistatic interactions of SNPs, in a large number of SNPs, is a computationally challenging task. We are, therefore, motivated to develop efficient and effective solutions for identifying epistatic interactions of SNPs.

Results: In this article, we propose an efficient Cloud-based Epistasis cOmputing (*eCEO*) model for large-scale epistatic interaction in genome-wide association study (GWAS). Given a large number of combinations of SNPs, our *eCEO* model is able to distribute them to balance the load across the processing nodes. Moreover, our *eCEO* model can efficiently process each combination of SNPs to determine the significance of its association with the phenotype. We have implemented and evaluated our *eCEO* model on our own cluster of more than 40 nodes. The experiment results demonstrate that the *eCEO* model is computationally efficient, flexible, scalable and practical. In addition, we have also deployed our *eCEO* model on the Amazon Elastic Compute Cloud. Our study further confirms its efficiency and ease of use in a public cloud.

Availability: The source code of eCEO is available at http://www.comp.nus.edu.sg/~wangzk/eCEO.html.

Contact: wangzhengkui@nus.edu.sg

Received on December 7, 2010; revised on January 25, 2011; accepted on February 14, 2011

1 INTRODUCTION

It is becoming increasingly important and challenging in genome-wide association study (GWAS) to identify single nucleotide polymorphisms (SNPs) associated with phenotypes such as human diseases (e.g. breast cancer, diabetes and heart attacks). Traditionally, researchers focused on the association of individual SNPs with the phenotypes. Such methods can only find weak associations as they ignore the genomic and environmental context of each SNP (Moore and Williams, 2009). However, SNPs may interact (known as *epistatic interaction*) and jointly influence the phenotypes. As such, there has been a shift away from the one-SNP-at-a-time approach toward a more holistic and significant approach that detects the association between a combination of multiple SNPs with the phenotypes (Moore *et al*., 2010).

In the meantime, the number of discovered SNPs is becoming larger and larger. For example, the dataset from the Hapmap project (Frazer *et al*., 2007) contains 3.1 million SNPs and the 1000 genome project (Durbin *et al*., 2010) provides ~15 million SNPs. From a computational perspective, it is very time consuming to determine the interactions of SNPs. Given n SNPs, the number of k-locus is ${}^{n}C_{k}=\frac{n!}{k!(n-k)!}$. This renders existing statistical modeling techniques (which work well for a small number of SNPs) (Park and Hastie, 2008; Wu *et al*., 2009, 2010; Yang *et al*., 2010) impractical. Likewise, techniques that enumerate all possible interactions (Wan *et al*., 2010; Zhang *et al*., 2010) are not scalable for a large number of SNPs. To reduce the computation overhead, heuristics (Park and Hastie, 2008; Wu *et al*., 2009, 2010) 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 that may have been filtered out. Therefore, a scalable and efficient approach becomes attractive for such a computationally intensive task in a large-scale GWAS.

A promising solution to the computation challenge is to exploit parallel processing. There are a variety of high-performance computing solutions. For example, in Ma *et al*. (2008), a tool is provided for processing single-locus and two-locus SNPs analyses using a supercomputer. However, it is not easy for researchers to rewrite their own programs on specialized hardware. As another example, in Greene *et al*. (2010), two-locus SNPs analysis is performed using the graphics processing units (GPU). However, this requires the users to understand the GPU architecture well to fully exploit the computation power of the GPU. Instead, we aim to develop a cloud-based solution which has a number of benefits. First, the MapReduce framework (available in most cloud services) offers high scalability, ease of programming and fault tolerance. Secondly, most software can be easily deployed on the cloud and made accessible to all. Thirdly, there are already low-cost commercially available cloud platforms [e.g. Amazon Elastic Compute Cloud (Amazon EC2)]. Fourthly, the pay-as-you-use model of such commercial platforms also makes them attractive

*To whom correspondence should be addressed.

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

for end-users who need not own, manage and (over-)provision any computational resources.

Finding significant epsitatic interactions of SNPs involves two major computational challenges:

(1) Given a large number of combinations of SNPs, how can we distribute them across multiple processing nodes effectively to achieve load balancing?

(2) Given a single combination of SNPs, how can we efficiently compute the significance of its association with the phenotype?

In this article, we propose an *e*fficient *C*loud-based *E*pistasis *C*omputing (*eCEO*) model to find statistically significant epistatic interactions. Our *eCEO* model is based on Google's MapReduce framework (Dean *et al*., 2004), and is implemented over Hadoop, an open-source equivalent implementation of the MapReduce framework. We develop solutions for the two computational challenges mentioned above. For the first challenge, we develop and study two approaches to distribute a large number of combinations of SNPs across processing nodes—a Greedy model and a Square-chopping model. For the second challenge, we adopt a Boolean operation approach, which is similar to the method used in Wan *et al*. (2010), and other optimizations.

We apply our solutions for determining significant interactions for two loci and three loci as well as retrieving the top-k most significant answers. As a first cut, we have adopted a brute force approach that examines all possible interactions among the SNPs. This ensures that we do not miss any statistically significant interactions. Our method can be easily extended to deal with heuristics approaches. We validate our proposed *eCEO* model on our local cluster of over 40 nodes and on a public cloud (i.e. Amazon EC2). Our results show that our *eCEO* model is efficient, and that the MapReduce framework can be effectively deployed for bioinformatics research such as the GWAS.

A preliminary version of this article appears in Wang *et al*. (2010). There, we developed the *CEO* model where we design the Greedy parallel distribution approach to tackle the first challenge and adopt a naive solution for the second challenge. Here, we extended *CEO* in several directions that significantly improves its performance. First, we design a Square-chopping parallel distribution approach for the first challenge. Secondly, we develop efficient solutions for the second challenge in our *eCEO* model. Our experimental study in this article shows that *eCEO* outperforms *CEO* by a wide margin. For example, the experiment result shows that the execution time for processing 500 000 SNPs in a 43-node cluster is reduced from around 25 to 30 days using *CEO* model to 9 h using *eCEO* model. Thirdly, our *eCEO* model supports four test statistics to measure the significance of the association between a combination of SNPs and the phenotype; and users can choose an appropriate one that meets their needs. Fourthly, we show how easy it is to use our model in a public cloud.

The rest of this article is organized as follows. Section 2 provides some background knowledge. In Sections 3, and 4, we present our solutions to address the two computational challenges for finding significant epistatic interactions. In Section 5, we report results of a performance study on our own cluster and Amazon EC2. We also present an efficient approach to retrieve the top-k most significant answers. Finally, we discuss and conclude this article in Section 6.

2 BACKGROUND

2.1 The MapReduce framework

MapReduce architecture: 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 task execution and communication with the JobTracker.

MapReduce computational paradigm: the MapReduce computational paradigm exploits parallelism by dividing a processing job into smaller tasks, each of which runs on a processing node. 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 `mappers`) on different nodes. The `mappers` process the data read from the file system and produce a set of intermediate results which are 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 $(k1, v1)$ that are read from data chunks. Through the `map` function, the input set of $(k1, v1)$ pairs are transformed into a new set of intermediate $(k2, v2)$ pairs. The MapReduce library will sort and partition all the intermediate pairs and pass them to the `reducers`.

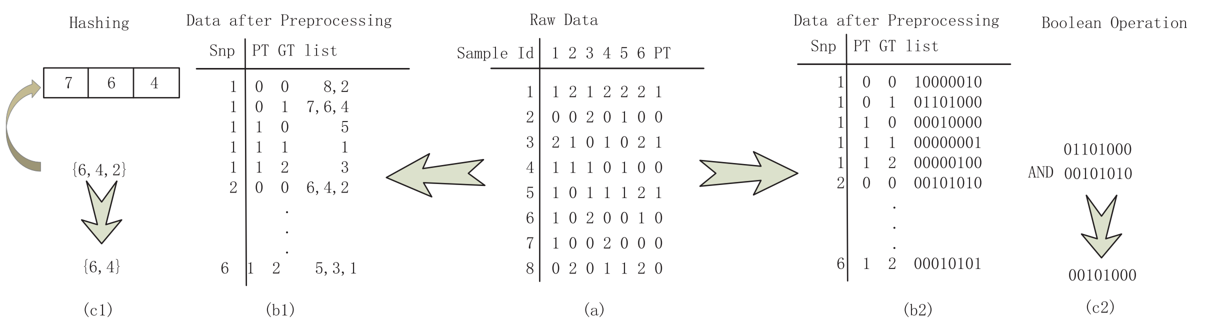
Shuffling phase: the partitioning function is used to partition the emitted pairs from the `map` phase into M partitions on the local disks, where M is the total number of `reducers`. The partitions are then shuffled to the corresponding `reducers` by the MapReduce library. Users can specify their own 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 operates on the key and values to produce a new $(k3, v3)$ pairs. The resultant output $(k3, v3)$ pairs are written back to the file system.

2.2 Statistical significance of SNP combinations

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 biallelic (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. Figure 1a shows an example of the raw data format for a dataset with eight individual samples and six SNPs. Each row contains the individual

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016



Snp	PT	GT	list
1	0	0	8, 2
1	0	1	7, 6, 4
1	1	0	5
1	1	1	1
1	1	2	3
2	0	0	6, 4, 2
.			
.			
.			
6	1	2	5, 3, 1

Sample Id	1	2	3	4	5	6	PT
1	1	2	1	2	2	2	1
2	0	0	2	0	1	0	0
3	2	1	0	1	0	2	1
4	1	1	1	0	1	0	0
5	1	0	1	1	1	2	1
6	1	0	2	0	0	1	0
7	1	0	0	2	0	0	0
8	0	2	0	1	1	2	0

Snp	PT	GT	list
1	0	0	10000010
1	0	1	01101000
1	1	0	00010000
1	1	1	00000001
1	1	2	00000100
2	0	0	00101010
		.	
		.	
		.	
6	1	2	00010101

Fig. 1. (a) The raw data format with six SNPs from eight individual samples; (b1) the data format after preprocessing with sample id list in *CEO* model; (c1) illustrates the hashing method for finding the intersection between two lists of sample ids in *CEO* model—one is sample id list from the SNP 1 whose PT and GT are 0 and 1, the other is the sample id list from the SNP 2 whose PT and GT are 0 and 0; (b2) the data format after preprocessing using bit strings representation in *eCEO* model; (c2) illustrates the way of finding the intersection from two lists with bit strings in *eCEO* model.

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.

The goal of our research is to identify a set of most significant combinations of multiple SNPs (epistatic interactions) that correlate to the phenotype. To measure the significance of the association between k-locus SNPs and phenotype in our *eCEO* model, we have implemented four test statistics namely, χ^2 test, likelihood ratio, normalized mutual information and uncertainty coefficient. Users can choose any of these based on their preferences and needs. As our *eCEO* framework assumes no statistical model fitting and is thus parameter free, these measures are effective in capturing interactions of arbitrary order. Due to space limitation, we shall focus on the χ^2 test (Balding *et al.*, 2006), which is used as the default in *eCEO*, in the following discussion.

Take two-locus 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, with these two groups of information, we can derive a 2×9 contingency table. We can calculate the χ^2 test value of this epistatic interaction from this contingency table using the following formula:

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

Where $n = \sum_{i=0}^{1}\sum_{j=0}^{2}\sum_{k=0}^{2} n_{i(j,k)}$, $n_i = \sum_{j=0}^{2}\sum_{k=0}^{2} n_{i(j,k)}$, 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. As the χ^2 test statistic follows the χ^2 distribution, the 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 interaction can be obtained as $P(\mathrm{x}>\mathrm{C})$ where C is the χ^2 test value, and $P(\mathrm{x})$ 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.

3 EFFICIENT ALGORITHM FOR FINDING THE ASSOCIATION SIGNIFICANCE

For our scheme to work, the raw data has to be preprocessed to collect the single SNP information in a new data format (to facilitate MapReduce processing). The straightforward way, which is adopted in our CEO model (Wang *et al.*, 2010), is to reorganize the data into the format of $<SNP_i, PT, GT, list(sampleID)>$ 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 dataset whose phenotype and SNP genotype on the SNP_i are PT and GT, respectively. Figure 1(b1) depicts the transformed data from the raw data in Figure 1a. The single SNP information can be sent to different processing nodes to calculate the χ^2 test value by collecting the contingency table from the combination of multiple SNPs. Let us still take two locus as an example. In order to collect the contingency table, the first step is to calculate the $n_{i(j,k)}$ from the single SNP information. If we want to calculate the $n_{i(j,k)}$ for the pair of SNP x and SNP y, we need the information from $<x, i, j, list1(sampleID)>$ in SNP x and $<y, i, k, list2(sampleID)>$ in SNP y. We can derive $n_{i(j,k)}$ from the intersection between the two sample id lists. This can be easily done as follows: first, we build a hash table for the sample ids in the first list; secondly, we use the sample ids in the second list to probe the hash table for matching sample ids. For example, to get $n_{0(1,0)}$ for the pair of SNP 0 and SNP 1, we intersect the two sample lists as shown in Figure 1(c1). However, our preliminary study suggests that using such an approach to collect the contingency table is computationally expensive.

In our *eCEO* model, we adopt an alternative approach. Instead of storing the sample ids in the list, we use a n-bit bit string to capture the sample ids, where n is the total number of samples. Each position in the bit string corresponds to a sample id. For example, from right to left, the first bit in the bit string corresponds to the sample with id 1, the second bit corresponds to the sample with id 2 and so on. If the sample id is in the list, its corresponding position will be set to 1, otherwise, it will be set to 0. Therefore, in our system, the new transformed data are represented as bit strings as in Figure 1(b2). With such a representation, we can perform an AND operation on the two bit strings to find the intersection between them more efficiently. We can easily get the number of intersection samples from counting the 1's bits from the AND result. Figure 1(c2) depicts calculating $n_{0(1,0)}$ for the pair of SNP 0 and SNP 1 using bit strings. This method provides a more cpu-efficient way of collecting the contingency table.

To further improve performance, we incorporate several optimizations in our *eCEO* model. (i) We use the mutable decoding scheme in our system. From our observation, immutable decoding of objects from the key/values into Java objects, used in the *CEO* model, is a time-consuming operation since it needs to create a unique Java object for each object in the key/values. For example, parsing 10 objects in each record for one million records needs to generate ten million immutable objects! With mutable decoding scheme,

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

Snp_id	PT	GT	Bit String
J	1	1	K

Fig. 2. Data format in bytes. J, 1, 1, K bytes are used to store the SNP ID, phenotype, genotype and the bit string of the sample id list. User can choose the value of J and K according to their data size.

we can reuse 10 mutable Java objects. Therefore, no matter how many records are decoded, only 10 objects are created and reused. (ii) We store all the data information into bytes including the SNP id, genotype, phenotype and bit strings as shown in the Figure 2. This follows from our observation that it is time consuming to use Java string split function to split the objects in a record. We parse the objects in a record by directly fetching from the bytes records. (iii) We write our own algorithm to count the intersection of the 1's bits without using the Java API.

4 PARALLEL DISTRIBUTION MODEL

4.1 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 2.2) to determine its significance. For N SNPs in the dataset, we need to calculate $\frac{N(N-1)}{2}$ two-locus SNPs combinations, as depicted in Figure 3a. 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: (i) how do we split the SNP pairs across all nodes? (ii) how to perform two-locus analysis under the MapReduce framework?

We shall first look at issue (i). Given N SNPs and M `reducers`, a naive 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 3a 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 metadata, 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. However, such a naive solution will result in load imbalance as some `reducers` are more heavily loaded than others, e.g. `reducer` one is likely to be a bottleneck. To achieve better load balancing, we propose two load-balanced solutions in this article:

Greedy model: 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 3a, 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. From our experimental results, we can see that our Greedy model has almost linear speed up when adding more resources which shows that our Greedy model is nearly load balanced.

Square-chopping model: under the Greedy model, the granularity of distribution of computation pairs is a single row. In some cases, if users have plenty of resources to use, they may want to reduce the number of computation pairs in each reducer further. Our Square-chopping model, which is an adaptation of the scheme in Ma *et al.* (2008), can be used in these scenarios. Instead of sending the combination pairs according to rows, we distribute them by 'square chopping' as shown in Figure 3a. This can be achieved by dividing N SNPs into m subsets evenly. Each subset has n SNPs where n equals $\frac{N}{m}$. For simplicity, n is assumed to be integer. Then we assign any two subsets into one reducer. As shown in Figure 3a, each off-diagonal reducer receives n^2 combination pairs and each diagonal reducer receives $\frac{n(n+1)}{2}$ combination pairs. Therefore, this scheme needs $\frac{m(m+1)}{2}$ reducers.

We are now ready to look at issue (ii), i.e. performing two-locus analysis in MapReduce framework. Without loss of generality, 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 3b shows how the *eCEO* model processes the data having six SNPs.

Map phase: each `mapper` reads a chunk of the input (preprocessed) data. For each SNP, it then determines the `reducers` with 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 3a, 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 3b subgraph (2), and `value` contains the rest of the SNP information including the genotype, phenotype and the bit string representing 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 3b 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 $i < j$. In the Greedy model, 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. For the Square-chopping model, this guarantees that information from one subset of SNPs will be supplied earlier than another. Therefore, we only need to keep one subset of the SNPs information in memory. As such, our *eCEO* model significantly reduces the memory utilization.

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

4.2 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 3a, we can get all the needed two-locus SNP 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 4 shows an example of finding all the three-locus SNPs with SNP_1 using the two-locus SNPs information from the first row in six SNPs example. In the same way, all the possible three-locus SNPs involving SNP_m can be generated from the

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

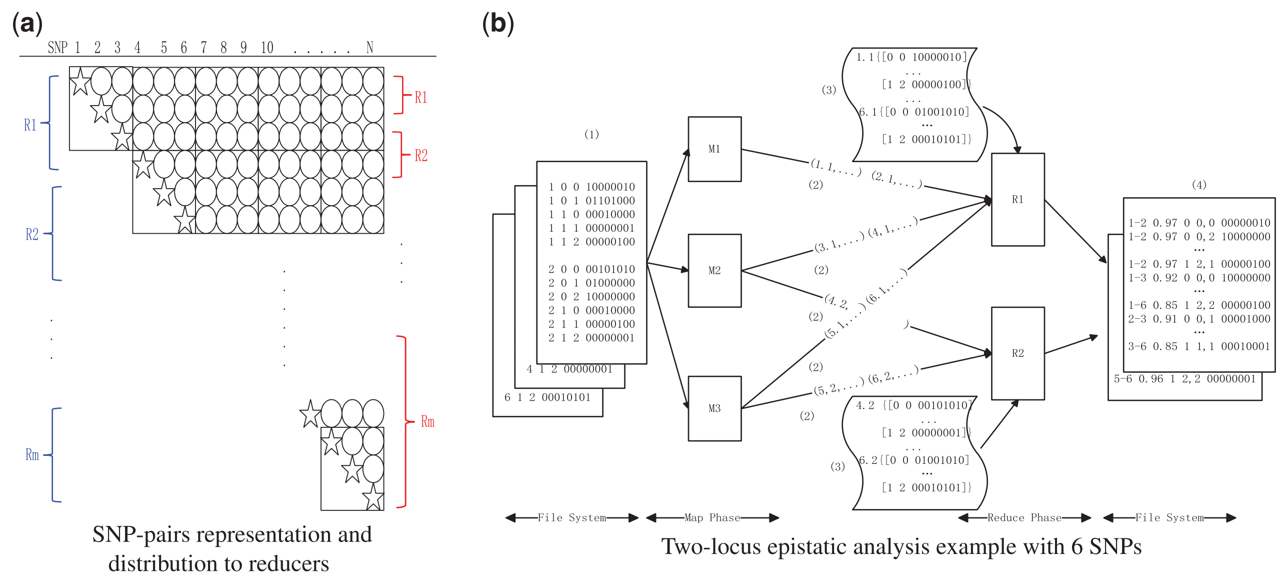

(a)
SNP 1 2 3 4 5 6 7 8 9 10 . . . . . N
R1
R2
Rm
SNP-pairs representation and distribution to reducers
(b)
(1)
(2)
(3)
(4)
M1
M2
M3
R1
R2
File System
Map Phase
Reduce Phase
File System
Two-locus epistatic analysis example with 6 SNPs

Fig. 3. Parallel distribution models and example of two-locus epistatic analysis using *eCEO* model.

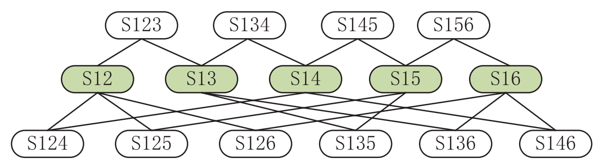


Fig. 4. All the three-locus SNPs having SNP1.

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 are 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 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 the `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.

5 RESULTS

Apache Hadoop is an open-source equivalent implementation of the MapReduce framework, running on Hadoop distributed file system (HDFS). We conduct a series of experiments on our local cluster with over 40 nodes, and a public cloud environment, Amazon Elastic Compute Cloud (Amazon EC2). For our local cluster, each node consists of a aX3430 4(4) @ 2.4 GHz CPU running Centos 5.4 with 8 GB memory and 2x 500G SATA disks. For Amazon EC2, we use 20 extra large instances, each with 8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each), 15 GB of memory and 1690 GB of local instance storage running on a 64-bit platform. 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 in our local cluster and 8 in EC2 instances. This guarantees that each `reducer` can get one core. Therefore, there are a total of 4*N and 8*N `reducers` which can be run simultaneously on a N-node local cluster and EC2 clusters, respectively.

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 one phase; with 360 `reducers`, it will then take three 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 50 000 SNPs dataset on a local 30-node cluster. Note that all the datasets we used include 2000 samples.

Figure 5a presents the running time for the Greedy model. 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 that increases the communication

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

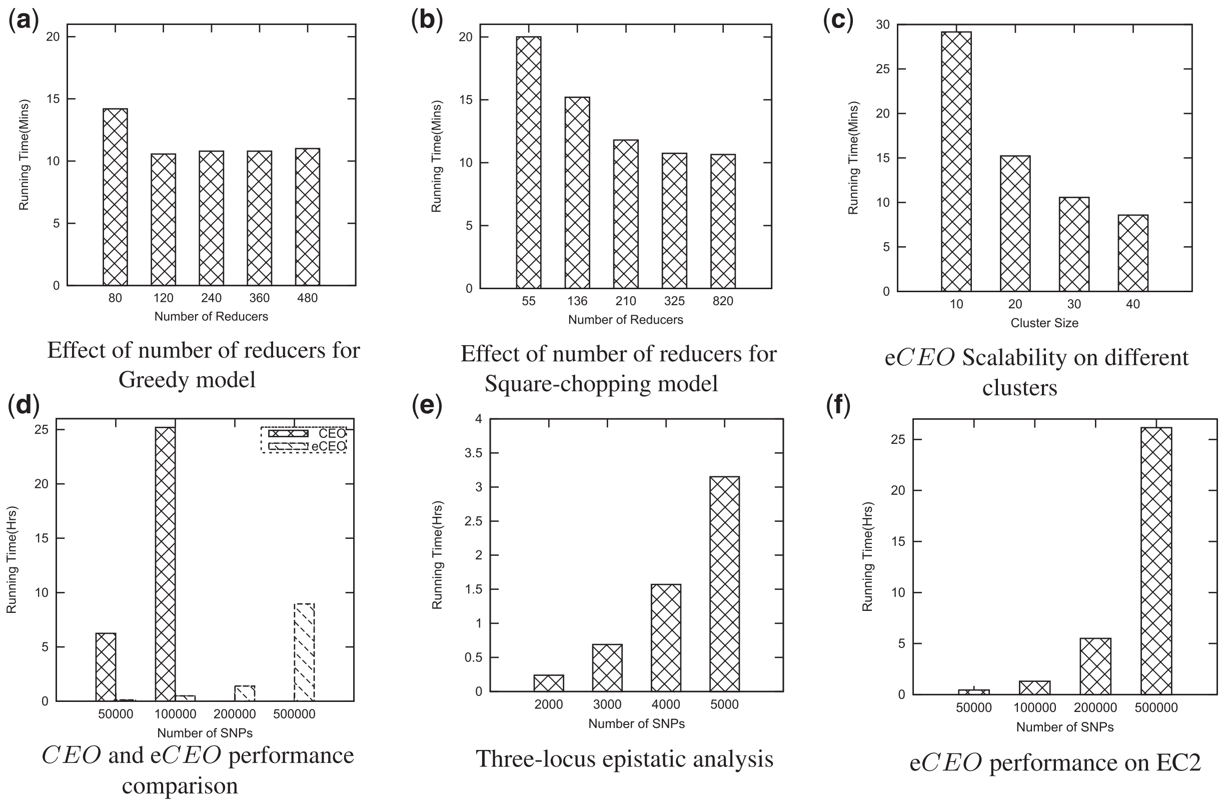


Fig. 5. (**a–d**) are the evaluation results for two-locus epistatic analysis on own local cluster. (**e**) The performance evaluation result for three-locus epistatic analysis on our own cluster. (**f**) The performance study on Amazon EC2.

overhead. We note that the Greedy model is optimal when the number of `reducers` corresponds to the actual configured value (i.e. 120).

Figure 5b presents the running time for the Square-chopping model. Here, the 50 000 SNPs are evenly split into 10, 16, 20, 25 and 40 partitions corresponding to 55, 136, 210, 325 and 820 reducers needed. From the results, we observe that when the reducer number is close to a multiple of *N*, where *N* is the total number of reducers configured in the cluster, its performance is good; otherwise ($N - R\%$N), reducers in the last phase, where R is the reducer number set in the job, will be wasted.

Looking at the results for the Greedy and the Square-chopping models, we observe that the Square-chopping model is generally inferior to the Greedy model. This is because of wasted reducers in the last phase (as discussed above). Its performance, however, is closer to the Greedy model as the partition number increases because the task in each reducer is smaller, and hence the wasted reducers in the last phase will not affect the total performance so much. Based on these results, for the subsequent experiments, we only use the Greedy model.

Scalability: first, we study the scalability of the *eCEO* model as the system resources increase. Figure 5c shows the completion time to analyze 50 000 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 completion time reduces with increasing number of nodes. In fact, we observe a (almost) linear speedup in performance. When we double the resources, the execution time reduces by half.

Now, let us consider the scalability of *eCEO* as the number of SNPs increases. Figure 5d shows the processing time for exhaustively computing all the significant interactions for two locus with 50 000, 100 000, 200 000 and 500 000 SNPs on a local 43-node cluster, and output the results whose *P*-values are smaller than 0.05. We made two interesting observations. First, the result shows that our *eCEO* offers a feasible and practical solution to perform pairwise epistasis for a large number of SNPs. According to Ma *et al.* (2008), 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. Our eCEO model can accomplish this task in not more than 9 h using only a 43-node cluster. Second, we note that the processing time is essentially proportional to the number of interacting SNP pairs to be evaluated. For example, the number of SNP pairs for the 500 000 SNPs dataset is 100 times more than that for the 50 000 SNPs dataset, and 6 times more than that for the 200 000 SNPs dataset. The running time for the 500 000 SNPs dataset (~538 min) is no more than 100 times that of the 50 000 SNPs dataset (~7 min), and is ~5 times more than the 200 000 SNPs dataset (~109 min).

Performance comparison between CEO and eCEO models: we also evaluate CEO's scalability with respect to the number of SNPs. The result is shown in Figure 5d. Clearly, *eCEO* outperforms *CEO* by a wide margin. We did not run the experiments for 200 000 and

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

500 000 SNPs in the *CEO* model because it will take a long time—we estimated the execute time for 500 000 SNPs to be roughly 25–30 days. But our *eCEO* model only needs 9 h to complete. We expect our *eCEO* model to be able to process 1 million SNPs in around 10 h on a 200-node cluster. The results confirm that our various optimizations are effective and efficient, and that our *eCEO* model is a practical and effective solution for processing large number of SNPs.

Three-locus analysis: we also evaluate the performance of three-locus analysis on a 43-node local cluster. We output all the two-locus analysis result and then perform the three-locus analysis. The result is presented in Figure 5e for SNP sizes of 2000, 3000, 4000 and 5000. We observe that the running time is also proportional to the number of SNP triples. This confirms that the *eCEO* scheme can effectively balance the load across all nodes.

Top-k retrieval: in our system, we store the results 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*-values. 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 (with size of 56 GB) only takes 145 s in the 43-node cluster.

Evaluation on and experience with a Public Cloud: *eCEO* is developed with the intention for users to exploit cloud computing for epistasis analysis. As such, we also evaluate our Greedy model on a public cloud, namely, Amazon EC2. Our quota of using Amazon EC2 instances in our research grant is 20. We use 20 extra large instances in our experiments, including 1 master node and 19 slaves nodes. There are 19 computation nodes in this experiment. Figure 5f shows the execution time for two-locus analysis as we vary the number of SNPs from 50 000 to 500 000. From the results, we can see that the execution time is essentially proportional to the number of interacting SNP pairs as we observed in our local cluster.

Our experience with Amazon EC2 shows the ease in which we can deploy our *eCEO* model. In fact, in the Hadoop package that we use, it provides tools to launch Amazon EC2 cluster with Hadoop directly. Therefore, we do not need to make any changes to our code. We do not even need to set up Hadoop at all. Once we launch the cluster in Amazon EC2, we simply upload our *eCEO* program and run it. With many cloud providers offering services to use MapReduce program directly (such as Amazon EC2, Amazon Elastic MapReduce and so on), our *eCEO* model is an important tool for large-scale epistasis analysis on a public cloud.

6 CONCLUSION

This article aims at providing an efficient epistasis computing model for large-scale epistatic interaction in GWAS which can be run on a computing cluster (local or cloud-based). We have proposed an efficient and feasible solution, called *eCEO* based on the MapReduce framework. As such, *eCEO* inherits the nice properties of MapReduce, which is high scalability and good fault tolerance. Moreover, it can leverage cloud computing with almost unlimited elastic computing resources. We have demonstrated the practical advantage of using *eCEO* model to exhaustively search two-locus and three-locus epistatic interactions. Our *eCEO* model can also retrieve top-k most significant interactions. We have conducted extensive experimental study on a local cluster of over 40 nodes and 20 instances on Amazon EC2. The results showed that our *eCEO* model is computationally efficient, flexible, scalable and practical. As future work, we plan to implement more test statistics. We also plan to explore the possibility of integrating *eCEO* as a filtering step to other methods, e.g. those based on statistical model fitting. Finally, we plan to develop pruning strategies based on domain knowledge, and integrate these into our scheme. For example, by knowing that certain SNPs do not interact, their computations can be avoided totally.

ACKNOWLEDGEMENT

Z.W. and Y.W. are supported by scholarships from the NUS Graduate School for Integrative Sciences and Engineering. Amazon Web Services in Education provides a research grant that supports our experiments on Amazon EC2.

Conflict of Interest: none declared.

REFERENCES

Balding,D.J. (2006) A tutorial on statistical methods for population association studies. *Nat. Rev. Genet.*, **7**, 781–791.

Dean,J. and Ghemawat,S. (2004) MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, USENIX Association, pp. 137–150.

Durbin, R.M. *et al.* (2010) A map of human genome variation from population-scale sequencing. *Nature*, **467**, 1061–1073.

Frazer,K.A. *et al*. (2007) A second generation human haplotype map of over 3.1 million SNPs. *Nature*, **449**, 851–861.

Greene,C. *et al.* (2010) Multifactor dimensionality reduction for graphics processing units enables geneome-wide testing of epistasis in sporadic ALS. *Bioinformatics*, **26**, 694–695.

Ma,L. *et al*. (2008) Parallel and serial computing tools for testing single-locus and epistatic SNP effects of quantitative traits in genome-wide association studies. *BMC Bioinformatics*, **9**, 315.

Moore,J.H. and Williams,S.M. (2009) Epistasis and its implications for personal genetics. *Am. J. Hum. Genet.*, **85**, 309–320.

Moore,J.H. *et al.* (2010) Bioinformatics challenges for genome-wide association studies. *Bioinformatics*, **26**, 445–455.

Park,M.Y. and Hastie,T. (2008) Penalized logistic regression for detecting gene interactions. *Biostatistics*, **9**, 30–50.

Wan,X. *et al.* (2010) BOOST: a fast approach to detecting gene-gene interactions in genome-wide case-control studies. *Am. J. Hum. Genet.*, **87**, 325–340.

Wang,Z. *et al.* (2010) CEO: a Cloud Epistasis cOmputing model in GWAS. In *Proceedings of IEEE International Conference on Bioinformatics and Biomedicine*, IEEE Computer Society, pp. 85–90.

Wu,T.T. *et al.* (2009) Genome-wide association analysis by lasso penalized logistic regression. *Bioinformatics*, **25**, 714–721.

Wu,J. *et al.* (2010) Screen and clean: a tool for identifying interactions in genome-wide association studies, *Genet. Epidemiol.*, **34**, 275–285.

Yang,C. *et al*. (2010) Identifying main effects and epistatic interactions from large-scale SNP data via adaptive group lasso. *BMC Bioinformatics*, **11** (Suppl. 1), S18.

Zhang,X. *et al.* (2010) TEAM: efficient two-locus epistasis tests in human genome-wide association study. *Bioinformatics*, **26**, 217–227.

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016