1. **Why is MapReduce important in the context of big data?**

Because Modern data-mining applications require processing of large amounts of data quickly, and modern computing systems use computing clusters as opposed to supercomputers, therefore, MapReduce programs which run on a computer cluster is very suitable for this area. And MapReduce provides abstract operation and parallel programming interface which can easily complete large-scale data programming and calculation processing.

1. **Why can MapReduce be considered a form of parallel computing?**

Because many computations are conceptual straightforward. When data is large computation must be distributed. This leads to issues of how to parallelize computation,

distribute data, and handle failures. And MapReduce allows us to express these simple

computations but hides the messy details of: parallelization, fault-tolerance, data distribution and load balancing.

1. **Describe some properties of MapReduce.**

Mapper: Accept a key-value pair and generate a set of intermediate key-value pairs. The MapReduce framework will pass the same value in the intermediate key-value pair generated by the mapper function to a reducer function.

Reducer: Accept a key and a related set of values, and combine the set of values to produce a set of smaller values.

1. **Describe the MapReduce programming model.**

User specifies the map and reduce functions.

Map function applied to each logical "record" in the input.

Returns set of intermediate key/value pairs.

All intermediate values associated with the same intermediate key are grouped.

Reduce function applied to all the values that share the same key.

Returns set of key/value pairs.

1. **Provide pseudocode for a MapReduce algorithm which computes the number of occurrences of each word in a large collection of documents.**

function map(String name, String document):

// name: document name

// document: document contents

for each word w in document:

emit (w, 1)

function reduce(String word, Iterator partialCounts):

// word: a word

// partialCounts: a list of aggregated partial counts

sum = 0

for each pc in partialCounts:

sum += pc

emit (word, sum)