# CS 61C Fall 2020 — MR, Spark, WSC, RAID, ECC

Discussion 14: November 30, 2020

## 1 Pre-Check

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 MapReduce is more general than Spark since it is lower level.

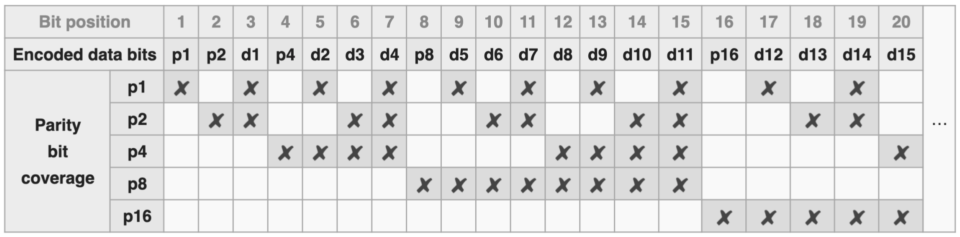
1.2 The higher the PUE the more efficient the datacenter is.

1.3 Hamming codes can detect any type of data corruption.

1.4 All RAID levels improve reliability.

## 2 Hamming ECC

Recall the basic structure of a Hamming code. We start out with some bitstring, and then add parity bits at the indices that are powers of two (1, 2, 8, etc.). We don't assign values to these parity bits yet. **Note that the indexing convention used for Hamming ECC is different from what you are familiar with.** In particular, the 1 index represents the MSB, and we index from left-to-right. The $i$th parity bit $P\{i\}$ covers the bits in the new bitstring where the *index* of the bit under the aforementioned convention, $j$, has a 1 at the same position as $i$ when represented as binary. For instance, 4 is `0b100` in binary. The integers $j$ that have a 1 in the same position when represented in binary are 4, 5, 6, 7, 12, 13, etc. Therefore, $P4$ covers the bits at indices 4, 5, 6, 7, 12, 13, etc. A visual representation of this is:



| Bit position | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Encoded data bits | | p1 | p2 | d1 | p4 | d2 | d3 | d4 | p8 | d5 | d6 | d7 | d8 | d9 | d10 | d11 | p16 | d12 | d13 | d14 | d15 | |
| Parity bit coverage | p1 | ✗ | | ✗ | | ✗ | | ✗ | | ✗ | | ✗ | | ✗ | | ✗ | | ✗ | | ✗ | | |
| | p2 | | ✗ | ✗ | | | ✗ | ✗ | | | ✗ | ✗ | | | ✗ | ✗ | | | ✗ | ✗ | | ... |
| | p4 | | | | ✗ | ✗ | ✗ | ✗ | | | | | ✗ | ✗ | ✗ | ✗ | | | | | ✗ | |
| | p8 | | | | | | | | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | | | | | | |
| | p16 | | | | | | | | | | | | | | | | ✗ | ✗ | ✗ | ✗ | ✗ | |

**Source:** `https://en.wikipedia.org/wiki/Hamming_code`

2.1 How many bits do we need to add to $0011_2$ to allow single error correction?

2.2 Which locations in $0011_2$ would parity bits be included?

2.3 Which bits does each parity bit cover in $0011_2$?

2.4 Write the completed coded representation for $0011_2$ to enable single error correction. Assume that we set the parity bits so that the bits they cover have even parity.

2.5 How can we enable an additional double error detection on top of this?

2.6 Find the original bits given the following SEC Hamming Code: $0110111_2$. Again, assume that the parity bits are set so that the bits they cover have even parity.

2.7 Find the original bits given the following SEC Hamming Code: $1001000_2$

## 3 RAID

3.1 Fill out the following table:

| | **Configuration** | **Pro/Good for** | **Con/Bad for** |
|---|---|---|---|
| RAID 0 | | | |
| RAID 1 | | | |
| RAID 2 | | | |
| RAID 3 | | | |
| RAID 4 | | | |

| | | | |
|---|---|---|---|
| RAID 5 | | | |

## 4 MapReduce

For each problem below, write pseudocode to complete the implementations. Tips:

- The input to each MapReduce job is given by the signature of `map()`.
- `emit(key k, value v)` outputs the key-value pair `(k, v)`.
- **`for`** `var in list` can be used to iterate through `Iterables` or you can call the `hasNext()` and `next()` functions.
- Usable data types: **`int`**, **`float`**, `String`. You may also use lists and custom data types composed of the aforementioned types.
- `intersection(list1, list2)` returns a list of the common elements of `list1`, `list2`.

4.1 Given a set of coins and each coin's owner in the form of a list of CoinPairs, compute the number of coins of each denomination that a person has.

```
CoinPair:
    String person
    String coinType
```

```
map(CoinPair pair):
```

```
reduce(________________, ________________):
```

4.2 Using the output of the first MapReduce, compute each person's amount of money. `valueOfCoin(String coinType)` returns a float corresponding to the dollar value of the coin.

```
map(tuple<CoinPair, int> output):
```

```
reduce(________________, ________________):
```

## 5 Spark

**Resilient Distributed Datasets (RDD)** are the primary abstraction of a distributed collection of items

**Transforms** $RDD \to RDD$

- `map`($f$) Return a new transformed item formed by calling $f$ on a source element.
- `flatMap`($f$) Similar to map, but each input item can be mapped to 0 or more output items (so $f$ should return a sequence rather than a single item).
- `reduceByKey`($f$) When called on a dataset of $(K, V)$ pairs, returns a dataset of $(K, V)$ pairs where the values for each key are aggregated using the given reduce function $f$, which must be of type $(V, V) \to V$.

**Actions** $RDD \to Value$

- `reduce`($f$) Aggregate the elements of the dataset *regardless of keys* using a function $f$.

Call `sc.parallelize(data)` to parallelize a Python collection, `data`.

$\boxed{5.1}$ Given a set of coins and each coin's owner, compute the number of coins of each denomination that a person has. Then, using the output of the first result, compute each person's amount of money. Assume `valueOfCoin(coinType)` is defined and returns the dollar value of the coin.

The type of `coinPairs` is a tuple of (person, coinType) pairs.

```
coinData = sc.parallelize(coinPairs)
```

$\boxed{5.2}$ Given a student's name and course taken, output their name and total GPA.

```
CourseData:
    int courseID
    float studentGrade // a number from 0-4
```

The type of `students` is a list of (studentName, courseData) pairs.

```
studentsData = sc.parallelize(students)
```

## 6 Warehouse-Scale Computing

Sources speculate Google has over 1 million servers. Assume each of the 1 million servers draw an average of 200W, the PUE is 1.5, and that Google pays an average of 6 cents per kilowatt-hour for datacenter electricity.

$\boxed{6.1}$ Estimate Google's annual power bill for its datacenters.

$\boxed{6.2}$ Google reduced the PUE of a 50,000-machine datacenter from 1.5 to 1.25 without decreasing the power supplied to the servers. What's the cost savings per year?