Advanced computer architecture

Adam Boult (www.bou.lt)

February 10, 2023

Contents

Preface

This is a live document, and is full of gaps, mistakes, typos etc.

Part I

Parallel programming

Chapter 1

Parallel processing

1.1 Parallel processing

1.1.1 MapReduce

Framework for parallising problems.

Say we have text and we want to do a word count. We can split the text into multiple subsections, do a word count on each and add these up.

MapReduce can do this.

1: Splitting

The main server splits the database into subsections.

For example, the text document could be divided into M chunks, to be split across different servers.

The master server can then pass data to be Mapped when they have capacity. Each server could run Map on a large number of documents.

Each document here is stored as a key value pair. The value here would be the text, and the key would identify the document.

$\{2 : "AppleAppleCarrot"\}$

2: Mapping

Rather than simply doing a word count on the subsections, we do a more complex, but more efficient approach.

We map the key value pair to a list of key value pairs. In our example we would map this to identifiers for each word.

The output here would be

$[\{"Apple" : 1\}, \{"Apple" : 1\}, \{"Carrot" : 1\}]$

The Map function will differ depending on the use case.

3: Shuffling

We now want to combine the key value pairs. There are R keys from the intermediate output of the Map program.

For each of the R keys, the master server assigns a server to reduce the key.

The master server then notifies the Reduce worker of the location of each relevant document output from the Map process. The Reduce server then copies the key value pair.

The Reduce worker now has an input of key value pairs. In our example one would have received:

$\{"Apple" : 1\}$

$\{"Apple" : 1\}$

It would also have received other instances from other servers.

The Reduce worker can then combine these to form the following:

$\{"Apple" : [1, 1]\}$

4: Reduce

Once all the data has been mapped, the data can be reduced.

In our example this involves creating:

$\{"Apple" : 2\}$

The Reduce function will differ depending on the use case.

5: Output

The outputs of each reduce can then be shared back to the main server.

We now have our word count.

Part II

RAID

Chapter 2

Redundant Array of Independent Disks (RAID)

2.1 Introduction

2.1.1 Introduction

Take physical disk drives, create logical disk drives.

2.1.2 Striping

A single file is spread over multiple disks.

Can be done at bit/byte/block level.

2.1.3 Mirroring

Same data on multiple drives

2.1.4 Parity

Used for error detection.

In protocol for sending/saving we say that all bits must be even (eg 1100) (or odd)

For given bit of info, we add parity bit to guarantee that bit is indeed even/odd. 1100 becomes 11000; 1000 becomes 10000

Cannot correct errors, just detect them. only detects if odd number of errors.

Parity can be stored on dedicate disk, distributed.

alternative to parity bit: hamming code

2.1.5 RAID levels

RAID 0: Uses striping across disks, but no redunency. allows for improved read/write times. if any drive fails, all fail RAID 1: Data written identically to two drives. read times increased, as with raid 0, due to mirroring. writing is slower. no parity or striping RAID 2: bit level striping and hamming code. rarely used RAID 3: rarely used. byte level stripping. Dedicated parity disk RAID 4: dedicated parity disk RAID 5: block level striping, distributed parity RIAD 6: double distributed parity

2.1.6 ZFS

RAID-Z: Under ZFC, similar to RAID 5

2.1.7 Off-site backups