Computer System Benchmarking

Design Document

Content Table

1. Introduction

1.1. Problem Statement

This assignment aims to teach you how to benchmark different parts of a computer system, from the CPU, Memory and Disk.

This assignment has been carried out on Chameleon testbed (https://www.chameleoncloud.org), by using KVM virtual machine m1.medium (2 virtual processors with 4GB RAM and 40GB disk).

1.2. System Introduction

This document provides a detailed design of the Benchmarking Assignment.

In this assignment we have implemented a benchmarking program that covers the four components listed below in a series of separate experiments:

1. **CPU:**

   Language: C
   Threads: 1, 2, 4 & 8
   Calculation: GFlops & IOps

   This benchmarking is implemented in **C** programming language by using Multithreading & by calculating GFlops & IOps.

2. **Memory:**

   Language: C
   Threads: 1, 2, 4 & 8
   Block Size: 8B, 8KB, 8MB & 80MB
   Operations: Sequential Write, Random Write & Read+Write

   This benchmarking is implemented in **C** programming language by using **Multithreading, strncpy()** & **memcpy()** functions.

3. **Disk:**

   Language: C
   Threads: 1, 2, 4 & 8
   Block Size: 8B, 8KB, 8MB & 80MB
   Operations: Sequential Read, Random Read & Read+Write

   This benchmarking is implemented in **C** programming language by using **Multithreading, fwrite()** & **fread()** functions.

2. Architecture

2.1. CPU

CPU Benchmarking is done by using C programming language. In this benchmarking metrics used for measuring CPU performance are Giga FLOPS & Giga IOPS. GFLOPS measures processor speed in terms of double precision floating point operations per seconds, while GIOPS measures processor speed in terms of double precision integer operations per seconds.

To achieve parallelism we have implemented above operations by using Multithreading (POSIX) We used 1, 2, 4 & 8 threads to measure performance of CPU. To calculate exact time in Nano seconds we used 'gettimeofday ()'.

2.2. Memory

Memory Benchmarking is done by using C programming language. In this benchmarking metrics used for measuring Memory performance are Throughput in terms of Mega Bytes per Seconds and Latency in terms of Microseconds. Operations carried out for measuring throughput & Latency are Read/Write, Sequential Write and Random Write.

To achieve parallelism we have implemented above operations by using Multithreading (POSIX) We used 1, 2, 4 & 8 threads to measure performance of Memory simultaneously for each operation.

2.3. Disk

Disk Benchmarking is done by using C programming language. In this benchmarking metrics used for measuring Disk performance are Throughput in terms of Mega Bytes per Seconds and Latency in terms of Microseconds. Operations carried out for measuring throughput & Latency are Read/Write, Sequential Read and Random Read.

To achieve parallelism we have implemented above operations by using Multithreading (POSIX) We used 1, 2, 4 & 8 threads to measure performance of Disk simultaneously for each operation. To calculate exact time in Nano seconds we used 'gettimeofday ()'.

3. Implementation

3.1. CPU

CPU benchmarking is implemented by passing integer functions to calculate IOPS & GFlops. GFlops is calculated by multiplying CPU speed with number of CPU cores. Then multiplying that result with product of CPU instruction per cycle & number of CPU per node. IOps is calculated by Dividing 1 by the sum of the average latency and the average seek time.

```
 end=clock();
 cpu_gflops=((double) (end-start))/CLOCKS_PER_SEC;
 printf("\n Time used = %Lf ",cpu_gflops);
 double flopscpu= 10000000/ (double)cpu_gflops;
 double gflopscpu= (double)flopscpu/100000000UL;
 printf("\n Gflops : %f",gflopscpu);
 start1=clock();

 for(i=0;i< nthreads;i++)
  {
   pthread_create(&thread[i],NULL,&iops,(void *)(long)nthreads);
  }
 for(i=0;i<nthreads;i++)
   {
   pthread_join(thread[i],NULL);
   }
 end1=clock();
 cpu_iops=((double) (end1-start1))/CLOCKS_PER_SEC;
 printf("\n Time used iops = %Lf ",cpu_iops);
 double iops= 10000000/ (double)(cpu_iops);
 double giopscpu= (double)iops/100000000UL;
 printf("\n Iops : %f",giopscpu);


end2=clock();
long double Total_time_used=((double) (end2-start2))/CLOCKS_PER_SEC;
printf("\n Time total used = %Lf ",Total_time_used);
}
```

3.2. Memory

Memory benchmarking is implemented by simply generating file of 400 MB. After creating file Sequential write operation is carried out by using strncpy(). Random write operation is also done by using strncpy() but simultaneously at different positions. Finally Read+Write operation is implemented by using strncpy() & memcpy().

3.2.1. Read & Write

```
void *read_write()
{

    long r, w;
    pthread_mutex_lock( &mutex1 );

    for(r = 0; r < load_div; r += buffer_size)
        memcpy(char2, char1, buffer_size);

    for(w = 0; w < load_div; w += buffer_size)
        strncpy(char3, char2, buffer_size);

    pthread_mutex_unlock(&mutex1);
}
```

3.3. Disk

Disk benchmarking is implemented by Creating file by Read+Write operation by using fwrite() along with fread(), then Sequential Read operation is carried out by using fread(). Random Read operation is carried out by using fread() but simultaneously at different positions.

3.3.1. Read + write

```
void *read_write()
{
    long j,k;

    pthread_mutex_lock( &mutex1 );


  FILE *f = fopen("Disk_Benchmark.txt","w+");

  size_div = ((max_size/buffer_size)/th);

  for(j=0;j<size_div;j++)

{
  fwrite(char1,buffer_size,1,f);

}

fseek(f, 0, SEEK_SET);


for(int k=0;k<size_div;k++)
{
    fread(char2,buffer_size,1,f);
  }

  fclose (f);

 pthread_mutex_unlock( &mutex1 );
}
```

4. Future Enhancements

- Better performance can be achieved by parallelization by avoiding deadlock i.e. synching each & every Thread properly.

- Memory & Disk performance can be increased by transferring big block of data at a time by handling segmentation error.

- Network performance can be increased by intelligently handling traffic in network .