BENCHMARKING TOOLS
(CS553-PA1)
TECHNICAL DESIGN DOCUMENT

Contents

1 Introduction 3

1.1 Purpose 3
1.2 Scope 3
1.5 Design Assumptions 3

2 Technical Overview 4

2.1 CPU Benchmark 4
2.2 Memory Benchmark 4
2.3 Disk Benchmark 5
2.4 Network Benchmark 6

1 Introduction

This document gives a detailed implementation of the benchmarking tools for CPU, Memory, Disk and Network on a cluster. The entire codebase is in C programming Language.

1.1 Purpose

The purpose of the document is to define the design aspects and the approach taken to implement each of the benchmarking tools

1.2 Scope

The scope of the document will baseline the design, technical and integration details/approaches for the implementation of the benchmarking tools

1.5 Design Assumptions

- All the compute nodes within the cluster have the same specifications

2 Technical Overview

This section should briefly introduce the system context and design, and discuss the background to the project.

2.1 CPU Benchmark

Implemented a simple multithreaded program in C. The algorithm is as follows –

a. Read input file to fetch the number of threads and precision
b. Based on the precision, call the appropriate function, passing the number of threads as parameter
c. Set the loop variable to 1trillion/(no. of threads * 10)
d. Measure and return the time taken to complete all 1 trillion operations
e. Gflops is calculated as 1trillion/total time taken
f. Write output to file

.

In step (c), I have divided by 10 since I have a total of 10 operations in my loop.

```
while( i<maxval)
 {
   x+y;
   x-y;
   x*y;
   x/y;
   x+1.4;
   x-1.98;
   y+1.76;
   y-1.54;
   i++;
}
```

2.2 Memory Benchmark

Implemented the following algorithm to perform the benchmark –

a. Read input file to fetch the pattern, block size and number of threads into a structure
b. Compare the access pattern and initialize the thread functions(RWR or RWS), passing the structure as parameter.
c. If block size is not 1
    I. Run memcpy 1gb/(block size * threads) times to copy 1 gb of data from source to destination.
    II. For sequential access, increment the loop variable by the block size for each iteration
    III. For random access, generate a random number between 0 and number of blocks and then perform memcpy.
    IV. Repeat step I 100 times

d. If block size is 1 –
    I. Run memcpy 100 million/(number of threads) times to copy 1 byte from source to destination
    II. For sequential access, increment the loop variable by the block size for each iteration
    III. For random access, generate a random number between 0 and number of blocks and then perform memcpy.
e. Measure execution time and calculate throughput.

The data structure is defined as follows –

```
typedef struct data{
  int  threads;
  int  block_size;

} t_data;
```

2.3 Disk Benchmark

Implemented the following algorithm to perform the benchmark –

a. Read input file to fetch the pattern, block size and number of threads into a structure
b. If access pattern is RR or RS, create a 10 gb file in /tmp folder
c. Compare the access pattern and initialize the thread functions(RS,RR,WS,WR), passing the structure as parameter.
d. If block size is not 1
    I. Open file in read/write mode
    II. Run read/write 1GB/(block size * threads) times to copy 10 GB of data from source to destination.
    III. For sequential access, increment the loop variable by the block size for each iteration
    IV. For random access, generate a random number between 0 and number of blocks and then perform read/write.

f. If block size is 1 –
    I. Open file in read/write mode
    II. Run read/write I million/(block size * threads) times to write/read 1GB.
    III. For sequential access, increment the loop variable by the block size for each iteration
    IV. For random access, generate a random number between 0 and number of blocks and then perform read/write.
g. For random access, generate a random number between 0 and number of blocks and then perform read/write.

2.4 Network Benchmark

Network Benchmark has two components, TCP & UDP. Each component was designed separately.

Multithreaded TCP Server –

1. Create a socket and bind to a port
2. Listen to connections
3. Accept a connection from client
4. If size is 2 bytes, wait
    a. As the client sends data, keep adding to memory and refresh as it reaches 1GB
    b. Once 100 Gb has been read by server, send a received ack to the client
5. If size is 1 byte, respond back with 1 byte

Multithreaded TCP Client –

1. Read input file and fetch block size and threads
2. If block size is not 1, initialize a thread and connect to client sending 2 bytes of data. This sends a signal to the client that the throughput experiment is being conducted. This threads stays alive and listens to the client response.
    a. Create threads and send data to client
    b. Once 100GB has been sent over the network, The server will send a signal to the thread that was created in step 2.
    c. Stop transmission of data and measure time
3. If block size is 1, create threads and send 1 byte to the server in a loop ranging from 0 to 1 million/threads.

UDP Server –

1. Create a TCP socket and bind to a port and listen to connections
2. Create a UDP socket and bind to another port
    a. Accept a connection from client
    b. If size is 2 bytes, wait
3. As the client sends data, keep adding to memory and refresh as it reaches 1GB
4. Once 100 Gb has been read by server, send a received ack to the client via the TCP connection
5. If size is 1 byte, respond back with 1 byte

Multithreaded UDP Client –

1. Read input file and fetch block size and threads
2. If block size is not 1, initialize a thread and connect to the TCP server created in the above step and send 2 bytes of data. This sends a signal to the client that the throughput experiment is being conducted. This thread stays alive and listens to the client response.
    a. Create threads and send data to client

b. Once 100GB has been sent over the network, The server will send a signal to the thread that was created in step 2.

c. Stop transmission of data and measure time

3. If block size is 1, create threads and send 1 byte to the server in a loop ranging from 0 to 1 million/threads.