Mini Project Report

On

Neural Network based parallel load computation in distributed system environment

Submitted by

Dhruvik - 171IT225

Rakesh Pavan - 171IT154

Yogesh Choubey - 171IT252

Neeraj - 171IT226

Under the Guidance of

Dr. Savitha S

Dept. of Information Technology,

NITK, Surathkal

Date of Submission: April 10, 2019

Dept. of Information Technology

National Institute of Technology Karnataka, Surathkal.

2018-2019

Abstract

Modern world technologies and problems require heavy computations and huge processing power to solve. Single system cannot be configured to such a degree of complexity. Thus people go for distributed and cluster systems and divide the problem (or load) into sub-problems and solve them in parallel. In this project we try to address an important problem involved in parallelizing a task, i.e to efficiently find an optimal distribution of the given computational load in a way that yields minimum processing time, making effective use of resources available.

We use a neural network based approach trained on ground truth generated by a divide & conquer based ternary search algorithm, to find an optimal way to distribute the given load on the basis of dynamic system parameters fed as features. A comparative performance study against some common existing methods show that our approach performs better in terms of effectiveness and efficiency.

Contents

1 Introduction

Modern world technologies and problems require heavy computations and huge processing power to solve. Single system cannot be configured to such a degree of complexity. Thus people go for distributed and cluster systems and divide the problem (or load) into sub-problems and solve them in parallel.

Parallel programming has some advantages that make it attractive as a solution approach for certain types of computing problems that are best suited to the use of multiprocessors. The main reason for parallel programming is to execute code efficiently, since parallel programming saves time, allowing the execution of applications in a shorter wall-clock time.

As a consequence of executing code efficiently, parallel programming often scales with the problem size, and thus can solve larger problems. In general, parallel programming is a means of providing concurrency, particularly performing simultaneously multiple actions at the same time.

In this project we try to address an important problem involved in parallelizing a task, i.e to efficiently find an optimal distribution of the given computational load in a way that yields minimum processing time, making effective use of resources available.

2 Literature Review

In this section, we have presented work done by various researchers in the field of Effective Load Balancing in Distributed Systems and Message Passing Environments .

Paper [1] investigates three possible distributed solutions proposed for load balancing; approaches inspired by Honeybee Foraging Behaviour, Biased Random Sampling and Active Clustering. This has implications for many technical issues in Service Oriented Architectures and Internet of Services (IoS)-type applications; including fault tolerance, high availability and scalability.

Paper [2] investigates the method of distributing the packet to process the packet through a processor. The method also includes assigning the packet to a guest such that a distribution of the packet to the guest is based on an algorithm. The method further includes altering a first destination address of the packet to a second destination address. The second destination address may be based on a virtual network interface of the guest.

Paper [3] discusses the method and system of distributed load balancing. Load balancing

includes receiving, from a client, a connection request to establish a connection with a server; determining load balancing state information based at least in part on the connection request, and distributing the connection to a selected server among a plurality of servers according to a result of the RMC function.

Paper [4] presents an implementation of the load balancing mechanism using master-slave model and Berkeley Lab Checkpoint/Restart (BLCR) toolkit. The overall goal is to create a Master-Slave model through which we can migrate processes from highly loaded nodes to some dedicated lightly loaded nodes. The agent running on master node divides total work into equal subtasks and delegates these sub-tasks to several independent slave nodes. The master node computes the final result by aggregating the partial results returned by slaves.

Paper [5] introduces Partial Key Grouping (PKG), a new stream partitioning scheme that adapts the classical power of two choices to a distributed streaming setting by leveraging two novel techniques: key splitting and local load estimation. In so doing, it achieves better load balancing than key grouping while being more scalable than shuffle grouping. It also tests PKG on several large datasets, both real-world and synthetic.

Paper [6] categorizes and reviews the representative studies on task allocation and load balancing according to the general characteristics of varying distributed systems. This survey summarizes the general characteristics of distributed systems. Based on these general characteristics, this survey reviews the studies on task allocation and load balancing with respect to the following aspects: 1) typical control models; 2) typical resource optimization methods; 3) typical methods for achieving reliability; 4) typical coordination mechanisms among heterogeneous nodes and 5) typical models considering network structures.

2.1 Identified Gaps

We have observed few common problems with most of the existing systems that work on this task. These include :

(i) Not considering the system dynamics: Most of the earlier works don't consider the dynamics of the client systems (to which the load is being distributed) while trying to find an optimal distribution. System dynamics like the memory availability, processor usage etc in the client system can affect the overall performance to a large extent.

(ii) Not considering the load characteristics: Existing works don't consider the characteristics of the load while doing the parallel computation. Characteristic of a load is defined

as the nature of distribution of the time vs input size for the load. This can affect the overall performance.

(iii) Usage of deterministic methods : Deterministic algorithms and approaches based on complete search are effective but take up a lot of processing time. And they are very susceptible and sensitive to small changes and variations.

2.2 Problem Statement

Using Neural Networks trained on optimally generated data (using a special algorithm) to find an optimal distribution for the given load across the client systems, such that it yields minimum possible computation time.

2.3 Objectives

We wish to address the issues stated above by making use of few additional parameters. Machine Learning based models are very effective and are usually very robust against small changes or variations in the dynamics of the system. Few objectives we wish to address are :-

(i) Consider the system dynamics of all the client systems by scrapping the processor and memory usage details.

(ii) Consider the load characteristics by running a small instance of the load and getting key time stamp instances.

(iii) Getting a scaling ratio (a normalization) between the server and all the client systems by first using a dummy computation and recording the time stamp.

(iv) Using a Divide & Conquer based ternary search algorithm for efficiently finding an optimal distribution of the load, hence generating the ground truth to train the neural network.

(v) Using a neural network (trained on the above generated data) to predict an optimal distribution based on the current features/parameters.

3 Methodology

The methods used directly follow in line with the objectives listed above. Totally 149 features are used for prediction, which include 25x4 system parameters, and 50 time stamp instances in the server for understanding nature of the load.

3.1 Determining Load Characteristics

We determine the characteristics, i.e the nature of time plots vs the input size for the given load by running a small instance on the server side and recorded 50 time stamps. Machine Learning model can understand the nature of the loads during training by considering these time stamps also as additional features.

3.2 State of the client systems

Dynamics of the system are considered by scraping the system parameters from the 'top' command in Linux. Top command displays processor activity of Linux box and also displays tasks managed by kernel in real-time. It shows processor and memory are being used and other information like running processes. 24 key values are scrapped using a BASH script. We have also considered a extra dummy feature which is used to give an idea of the scaling/normalization between the clients and the server systems. The features are : load average 0, load average 1, load average 2, tasks, running tasks, sleeping tasks, stopped tasks, zombi tasks, % of cpu used us = Time spent in user space, sy = Time spent in kernel space, ni = Time spent running niced user processes (User defined priority), id = Time spent in idle operations, wa = Time spent on waiting on IO peripherals (eg. disk), hi = the time spent processing hardware interrupts, si = Time spent handling software interrupt routines. (a piece of code, calls an interrupt routine...), Kib memory total, Kib memory total free, Kib memory total used, Kib memory total buffer/cach, Kib swap total, Kib swap free, Kib swap used, Kib swap available memory, time unit = time for calculation of 50 (dummy feature)

3.3 Neural Network

We have used a neural network having a single hidden layer with 4 neurons, and activation set as relu activation. The input layer is a feature vector of 149 features, and the output

layer is a softmax distribution of size 4 which gives the optimal distribution for the given feature input.

4 Implementation

Client-server architecture can be considered as a network environment that exchanges information between a server machine and a client machine where the server has some resources that can be shared by different clients.

We used TCP. TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

Socket Programming This is the most widely used concept in Networking and provide a connection between server-client using sockets

Server:

The server instantiates a ServerSocket object, denoting which port number communication is to occur on. The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port. After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to. The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server. On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket. Client:

The java.net.Socket class represents the socket that both the client and the server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method. Using dataInputStream and dataOutputStream, the client can communicate with the server.

* DataOutputStream output = new DataOutputStream(clientSocket.getOutputStream());

* DataInputStream Input = new DataOutputStream(clientSocket.getInputStream());

4.1 Work Done

We generated dataset using client-server architecture. 4 client connect with one server and server distributes the given load. The load is generated by the server and distributed among them using 4 different percentage (a%,b%,c%,d%). We take the total time required and save it for data training. We create a different client-server architecture for taking data of TOP command and time required to calculate 50! for each client. According to this generated data file, we train our model to best distribute among all clients.

Equal distribution

The server connects with 4 different client architecture and it will start to distribute the load. Depending on the load each client responds with the required time to calculate the given load. The server connects with 4 clients with TCP/IP protocol. We used TCP/TP protocol to acknowledge whether the client is calculating the given load or not.

In Equal distribution, If the server has the LOAD, then it will distribute equally among all clients.

client0 : 0 - LOAD/4

client1: LOAD/4 + 1 - 2*LOAD/4

client2: 2*LOAD/4 + 1 - 3 *LOAD/4

client3 : 3 *LOAD/4 + 1 - LOAD

The time required for each system to calculate the give equally distributed load:

clint(i) = T(i) , 0¡= i ¡=3

so, the total time required for calculating equally distributed load is :

TimeEquallyDistribution = maxOf (T(0) , T(1) , T(2) , T(3))

Machine Learning Distribution

We have all the generated data using a different distribution of load. The load is generated by the server and distributed among them using 4 different percentage (a%,b%,c%,d%). This generated data is mapped with the minimum time to execute the load using the ternary search method with N*N *log(N) time complexity.

In ML distribution, If the server has the LOAD, then it will distribute according to the result of the ML model. The ml Model generates 4 ranges. According to the ranges, the server has to divide their load among the client.

for example, ML model generates 4 ranges r1,r2,r3,r4;

client0 : 0 - r1

client1: (r1+1) - (r2+r1)

client2: (r2+r1+1) - (r2+r1+r3)

client3 : (r2+r1+r3 + 1) - (r2+r1+r3+r4)

The time required for each system to calculate the give ML distributed load is:

clint(i) = TM(i) , 0¡= i ¡=3

so, the total time required for calculating equally distributed load is :

TimeMLDistribution = maxOf (TM(0) , TM(1) , TM(2) , TM(3))

after the result, we compare TimeMLDistribution with TimeEquallyDistribution

4.2 Divide & Conquer based Ternary Search

Finding optimal load distribution using ternary search (Generating ground truth). For the given task, ternary search is used to find out the most optimal load distribution(load distribution that minimizes the overall time for execution of the complete program on parallel processor). Let the load distribution be given as a function f(x, y, z, w), where x, y, z and w are the percentages of input to be calculated by the 4 systems and f(x, y, z, w) is the overall time taken. By fixing 2 parameters say x and y the graph of f forms a parabola on 2D plane with global minima occuring at the tip of parabola that corresponds to the z value and w is given by 100-(x+y+z). Ternary search is a divide and conquer algorithm that divides the range into 3 parts and keeps 2 pointers corresponding to two mid values. Depending on the values of mid the range is further reduced to smaller subpart and the algorithm keeps generating smaller subparts until the global minima is reached.

Algorithm:

1. Iterate over all possible values of x over range [0, 100]

2. Keeping x fixed iterate over all possible values of y over range [x, 100].

3. Keeping x and y fixed, apply the following search procedure for z:

(i) Keep low = y, high = 100.

(ii) Let mid1 = low + (high-low)/3, mid2 = low + 2*(high-low)/3.

(iii) Calculate t1 = getTime(x, y-x, mid1-y) and t2 = getTime(x, y-x, mid2-y). Here x, y-x and mid1-y or mid2-y are the percentages of load distributed among the first 3 systems.

(iv) if t1 ¿ t2, then make low = mid1, ie. ignore the part [lo, mid1) or make high = mid2, ie. ignore the part (mid2, high], because the global minima can't lie in these parts.

(v) Keep performing step (ii) until mid2 - mid1 ¿ 1, because the accuracy is taken to be 1%.

4. Take the minimum of all the t1 computed by fixing x, y and applying ternary search for the values of z.

The getTime function returns the time taken for the computation for the respective load distribution.

The minTime returned gives the time required corresponding to the most effective load distribution.

Time Complexity:

The overall time complexity of the algorithm is O(n$\hat{2}$logn) corresponding to n=100, and the accuracy taken to be 1

The optimal solution generated based on the above approach is run several times on different systems to generate data to train the ML model.

4.3 Results and Analysis

We compare our approach with the normal Equal distribution method. We gradually increase the workload (which is calculating Factorial under modulo, starting from x = 1000, and mod = 10000000009). Table 1 lists our results with varying loads for both our Machine Learning based approach trained on the ground truth, against the equal distribution method. We clearly see improvement in the total computation time by large margins.

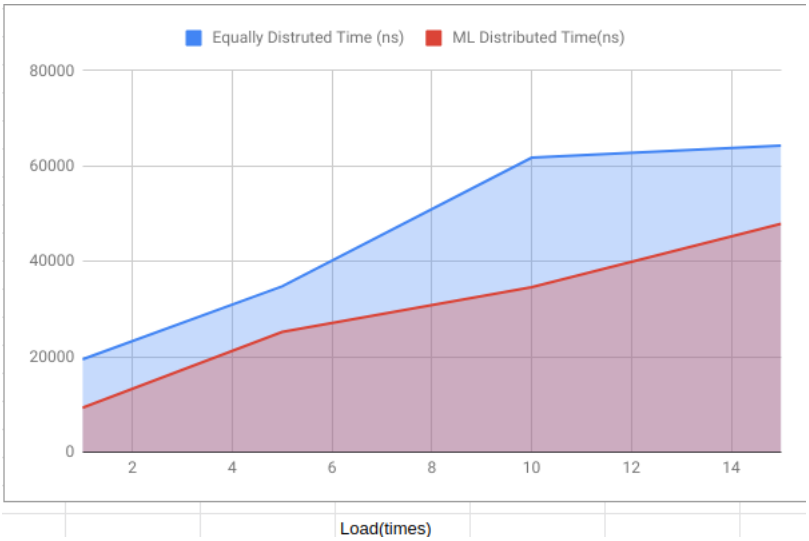

Figure 1: **Graph of Computation time vs Increasing load using our method against equal distribution, Y axis - Computation time in Nanoseconds, X axis - Load X times fact(1000)%mod**

Table 1: **Results showing percentage improvement, i.e reduction in computation time**

Load	Computation Time with our Method (ns)	With Existing Method (ns)	Percentage Improvement
1x	9289	19466	**52.280903%**
5x	25188	34777	**27.572823%**
10x	34574	61745	**44.005184%**
15x	47876	64277	**25.516125%**

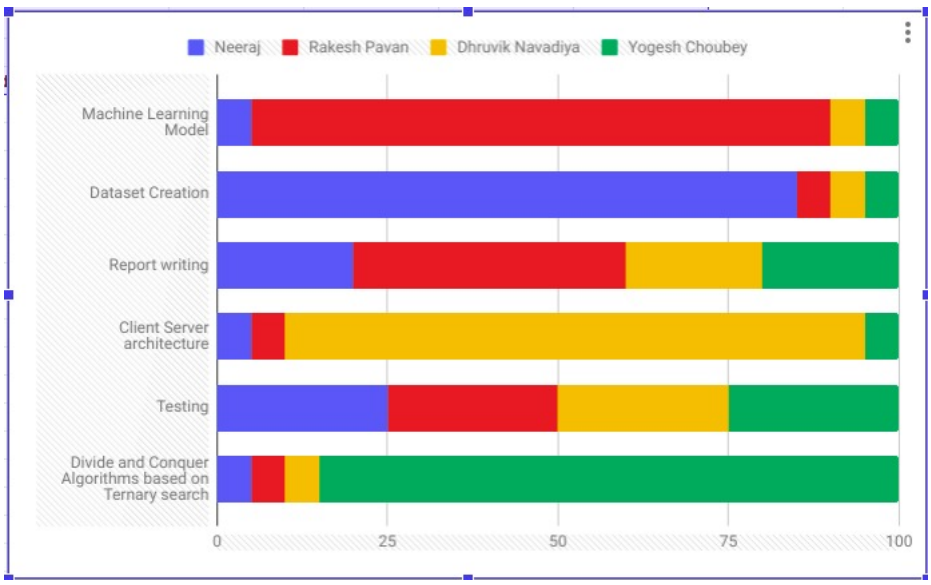

Figure 2: Gantt Chart Showing Individual Contribution

5 Conclusion and Future Trend

In this project we used a neural network based approach trained on ground truth generated by a divide & conquer based ternary search algorithm, to find an optimal way to distribute the given load on the basis of dynamic system parameters fed as features. A comparative performance study against some common existing methods showed that our approach performs better in terms of effectiveness and efficiency, giving reductions of **52.280903, 27.572823, 44.005184,** and **25.516125%** in total computation times for 1x, 5x, 10x, and 15x loads.

Future works can be extending this work for different kinds of loads, using other Machine learning algorithms, and using more system parameters for finding the optimal distribution for any kind of load.

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing.* 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops.

[2] Jianjun SHEN, Zhi Feng XIA, Mojiong QIU, Shudong Zhou, Donghai HAN. *Hypervisor level distributed load-balancing.* US patent App.

[3] Ranganathan Rajagopalan, Murali Basavaiah, Kiron Haltore, Anand Parthasarthy, Abhijeet Joglekar. *Method and system for distributed load balancing.* US patent App.

[4] Ravindra Anilkumar Vyas, Hardik Maheta,Vipul Dabhi, Harshadkumar B Prajapati. *Load balancing using process migration for linux based distributed system.* 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), Ghaziabad, India.

[5] Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, David Garca-Soriano, Nicolas Kourtellis. *The power of both choices: Practical load balancing for distributed stream processing engines.* 2015 IEEE 31st International Conference on Data Engineering.

[6] Yichuan Jiang. *A Survey of Task Allocation and Load Balancing in Distributed Systems.* IEEE Transactions on Parallel and Distributed Systems.