Implementing Discrete Cuckoo Search Algorithm for TSP using MPI and Beowulf Cluster

Bhavana V,Varshini Ramesh, Sivagami M

Abstract: The main aim of the proposed work is to find a better quality solution for Travelling Salesman Problem using Cuckoo Search Algorithm efficiently. Parallelizing is done to increase the quality of result. To implement parallelism across the nodes, MPI has been used. A Beowulf cluster was also created to achieve parallelism. The Discrete Cuckoo Search optimization code was run on a single system with multiple cores and on the Beowulf cluster. A thorough analysis has been performed to assess the efficacy of this optimization and the results are shown.

Index Terms*: *Discrete Cuckoo Search Optimization Algorithm, MPI, Travelling Salesman Problem

I. INTRODUCTION

A large number of problems in today's world are non-deterministic in nature. Many of these non-deterministic problems have NP-hard complexity and thus it is unknown if these problems are solvable in polynomial time and for a few of these problems it also cannot be verified in polynomial time. For these problems, optimization is used to find the 'best' solution.Optimization plays an important role in solving various engineering problems which are Non-deterministic in nature. The goal of the optimization process is to determine either a maximum or a minimum value of the problem being solved, generally known as the objective function. These problems include, but not limited to, systems design, electricity network operation, electricity generation, wireless communications routing and minimization of energy losses during electricity transmission. Proper measures of efficiency of optimization algorithms require assessment of computational time and convergence rate in addition to the accuracy to determine the minimum or maximum values.

Revised Manuscript Received on June 07, 2019

Bhavana V, Student, School of Computer Science Engineering, Vellore Institute of Technology, Chennai, India

Varshini Ramesh, Student,School of Computer Science Engineering, Vellore Institute of Technology, Chennai, India

Sivagami M, Associate Professor, School of Computer Science Engineering, Vellore Institute of Technology, Chennai, India

There are different types of optimization techniques. Brute force is systematically exhausting all possible solutions to find the optimum solution. Brute force often has high computational complexity and is unsuitable for large datasets. Greedy algorithm makes greedy choices at every step. It makes the choice that is best at that moment. This approach often gives only an average solution and not the optimal solution. The greedy algorithms fail especially when global optimal solution is required. Heuristics are used to solve and quickly find solutions by performing optimized search. When exact solutions are difficult to find, heuristics provide an alternative by providing approximate solutions quickly and are suitable for solving a large number of real-time problems. A few heuristic algorithms are Swarm intelligence, Genetic algorithms and Artificial neural networks. Meta-heuristics are a type of high level heuristics which are used to perform partial search and provide good optimal solutions when data is insufficient or computational capabilities are limited.Meta-heuristics are algorithmic frameworks used to develop optimization algorithms which are problem-independent and can be adapted to any problem statement. Meta Heuristicalgorithms are usually not problem specific and non-deterministic. Many of the meta-heuristic algorithms are inspired from and based on nature. The various meta-heuristic algorithms are used depending on the necessity and constraints of the problem. Swarm intelligence is a popular type of meta-heuristic algorithm, where a large number of individual objects interacting with each other are used to describe the collective behaviour of the system. It is used to find the optimal solution of various NP-hard problems. Examples of Swarm intelligence include Cuckoo Search Optimization algorithm (CSO), Particle swarm optimization (PSO), and Ant Colony optimization (ACO).The particle swarm optimization (PSO) is a stochastic algorithm where the list of potential solutions or 'particles' are moved in a 'search-space' by changing position and velocity. The particles movements are determined by the 'local best' and 'best known' positions which are constantly updated. The algorithm aims at guiding all the particles in the herd towards the best solutions.

Ant Colony Optimization algorithm (ACO) is inspired

ant colonies. 'Ants' move across the solution space to find the optimal solution. The ants continuously share their location and the quality of their solutions which allows more ants to find the better solutions closer to the optimal solution in later iterations. The larger the number of ants, better the solutions provided by the algorithm.

Cuckoo Search Optimization (CSO) is one such meta-heuristic algorithm which replicates brood parasitism found cuckoo's. The eggs of the bird represent the potential solutions. Gradually, the algorithm replaces the poor solutions with new and better solutions. The CSO has several applications in various fields such as neural networks, job scheduling, etc. One such problem which can be solved with Cuckoo Search Optimization is Travelling Salesman Problem.

The TravellingSalesmanProblem (TSP) is a combinatorial optimization problem. The Travelling Salesman Problem describes a salesman who must travel between N cities. The order of traversal is not considered but the salesman must travel to each of the nodes at least once and return to the starting node (city). Each of the edges (links between cities) has a corresponding weights (or the cost) attached. The cost for example, could be the cost of an airplane ticket or train ticket, or perhaps by the length of the edge, or time required to complete the traversal. The aim is to minimize cost as much as possible whilst performing traversal.

Since real-time data for TSP is large, time taken to process and perform computation is extremely large. To handle such large data, very high computational capabilities and power are required. These capabilities are found in supercomputers, which are uneconomical and difficult to access. A financially feasible solution to this problem is to use parallel and distributed computing.

Parallel computing is the type of computing where different parts of a single problem are compiled and executed simultaneously on different cores. Parallelism is done to improve the speed of a program. Distributed systems are a collection of systems which are interconnected. A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another.Cluster computing is a type of distributed system. A computer cluster is a set of connected computers that work together to perform a single task. They can be tightly coupled or loosely coupled.

II. LITERATURE SURVEY

A large number of meta-heuristic have been developed to solve various complex optimization problems, these meta-heuristic algorithms have often been inspired by and are based on natural occurrences'. Studies performed by Leonora Bianchi, et al (2009) show that Meta-heuristic algorithms are predominantly used overcome the disadvantages due to either incomplete information or limited computation capabilities [1]. Christian Blum, et al (2003) performed classification on meta-heuristic algorithms using various properties such as search strategy, whether they are single-solution or population-based, they can also be nature based meta-heuristics or parallel meta-heuristics [2]. The field of meta-heuristics for the application to combinatorial optimization problems is a rapidly growing field of research. This is due to the large number of combinatorial optimization problems which exist in scientific and the industrial world. The Travelling Salesman Problem is one such combinatorial optimization problem which was studied by E.L.Lawler (1985) and later by D.S.Johnson, et al(1997), to perform optimization on TSP, H.-C.Gao, et al (2006) studied various meta-heuristic algorithms such as ant colony algorithm, genetic algorithm, simulated annealing, tabu search, hopfield neural network, particle swarm optimization and immune algorithm, etc. then applied and identified the advantages and disadvantages of each algorithm [3]-[5]. Further research was performed by J.Ji, et al (2010) and Shin Siang Choong, et al (2019) using an ant colony based algorithm and artificial bee colony algorithm respectively [6]-[7]. Xin-She Yang, et al in 2009 developed the Cuckoo Search Optimization Algorithm which is a relatively new meta-heuristic search algorithm, inspired by the breeding behaviour and brood parasitism of cuckoos [8]. CS uses levy flights to perform random walk, which are useful in stochastic measurement and simulations for random or pseudo-random natural phenomena. R. N. Mantegna, et al (1994) first proposed accurate numerical simulation of Levy flights and further research was done by M. Gutowski (2001) on levy flights and their application in global optimization algorithms [9]-[10]. G.K.Jati, et al (2012) developed a Discrete Cuckoo Search Algorithm to solve the Travelling Salesman Problem, Later Aziz Quaarab, et al(2014) proposed an improved Discrete Cuckoo Search algorithm [11]-[12]. This algorithm has proved to be very efficacious in solving optimization problems. The performance of the proposed Discrete Cuckoo Search (DCS) is tested against a set of benchmarks of symmetric TSP from the well-kenned TSPLIB library. N.Tzy-Luen, et al(2016) developed a parallel Cuckoo Search algorithm for the Travelling Salesman Problem using OpenMP [13]. Similarly Arindam Majumdar, et al(2018) developed a parallel Cuckoo Search algorithm using batch processing [14]. In this proposed work, MPI has been used to implement Discrete Cuckoo Search Optimization Algorithm for TSP in a distributed environment.

This paper is organized as follows: Section 3 explains the proposed model; Section 4 explains Discrete Cuckoo Search Optimization; Section 5 contains the Implementation Details; Section 6 contains Results and Discussion, Section 7 Concludes the proposed work and Section 8 lists the Scope for Improvement.

III. PROPOSED MODEL

Travelling Salesman Problem will have a large real-time dataset.

Accuracy improves with the number of times the code is run for Swarm Intelligence based algorithms because they are based on random walk. This proposed model achieves better accuracy by running the algorithm parallelly on multiple cores/nodes and comparing those solutions to find the best optimized solution.

ACO is not efficient for large scale data. When the dataset size is large in the case of ACO, individual ants will move randomly, which will make the search for an optimum solution longer. Hence ACO is not suitable for large-scale data. Also, choosing the right parameters is empirical, and if the appropriate parameters

Published By:
Blue Eyes Intelligence Engineering &
Sciences Publication

Retrieval Number H6711068819 /19©BEIESP

are not chosen, it will mostly lead to efficiency and poor convergence of the algorithm. ACO has weak local search ability.

PSO suffers from partial optimization. The algorithm can get trapped in local optimum, which will lead to non-optimal solution. Probability distribution changes by iteration in PSO, hence the algorithm will be not be efficient. PSO also has a weak local search ability.Hence, Cuckoo Search Optimization is chosen to find the optimum solution in this proposed work. Since the Travelling Salesman Problem is a combinatorial optimization problem and Cuckoo Search Optimization is designed to apply for continuous optimization problem, this research proposes Discrete Cuckoo Search Optimization for solving TSP. Cuckoo search is a non-deterministic algorithm. Therefore, getting one solution may not be the best solution. The algorithm needs to be run multiple times.

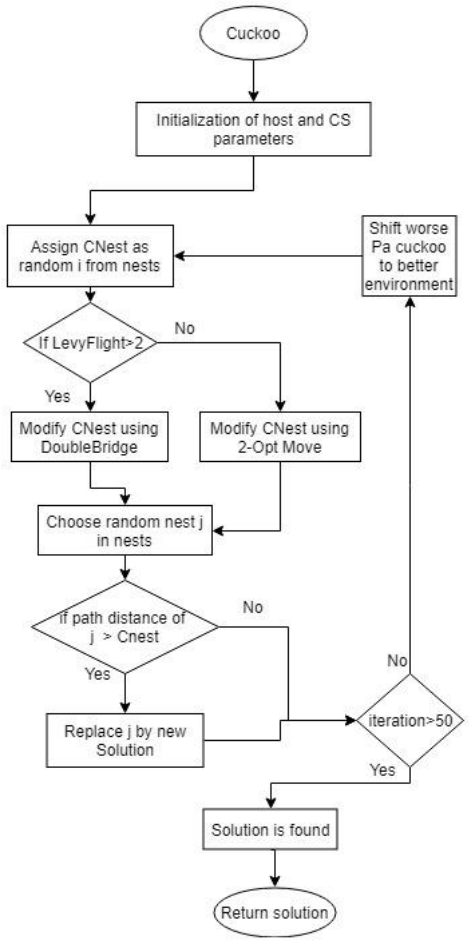


Fig. 1.a Cuckoo function

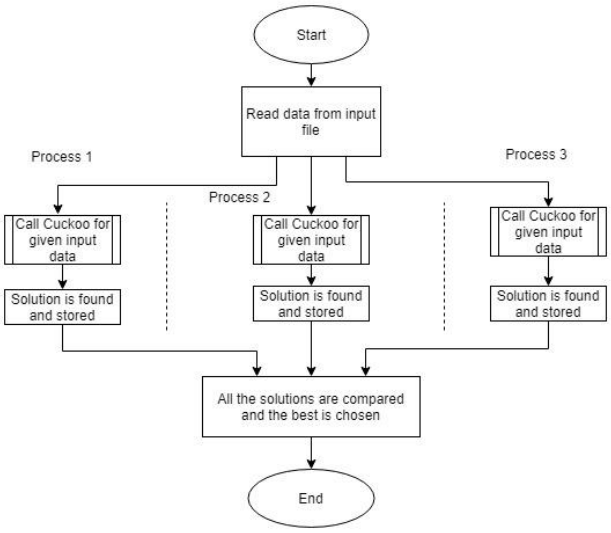


Fig. 1.b Cuckoo Search Optimization executed parallelly

IV. DISCRETE CUCKOO SEARCH OPTIMIZATION (DCSO)

Discrete Cuckoo Search Optimization algorithm uses random walks with Levy flights. Cuckoo Search is an Optimization algorithm developed by Xin-she Yang and Suash Deb in 2009. The algorithm was derived from nature of cuckoo species which lay their eggs in the nests of other birds. If the host bird finds alien eggs, i.e., cuckoo eggs, the host bird will throw these eggs or abandon the nest and create a new nest.

This breeding behavior is ideal and can be applied for optimization algorithms. This algorithm has the potential to outperform many other meta-heuristic algorithms.

In the Cuckoo Search Optimization an egg in a nest represents a solution, and a cuckoo egg represents a new and better solution. The final goal of the algorithm is to throw the bad solutions and replace them with better solutions. Each nest has one egg . Complex cases include multiple eggs in every nest.

Basic rules for the algorithm

1. Cuckoo can lay only one egg in an iteration, and discards its egg in a random nest.
2. Only the best solutions will carry on to the next generation.
3. Probability of the host bird finding the egg by the cuckoo is higher if the solution is bad. If an egg is detected, it is thrown to a faraway nest.

Cuckoo Search is used for continuous problems, and therefore we must use Discrete Cuckoo Search for solving TSP.

The Discrete Cuckoo Search Optimization is executed parallelly in each of the cores. The objective function used to optimize the TSP, minimizes the traversal cost of the path. A new random path is generated using Levy flights. The cost of this new random path and the cost of the existing path is calculated and compared. The path the higher cost is discarded and the path with lesser cost is retained. After each iteration, Pa percentage of the worst nests are discarded.

This program is run in ‘n’ cores parallelly, and ‘m’ random cities are selected.

Each node selects a different starting city and computes a traversal path. Randomizing the cities selected helps in a better optimized solution. Each node where this program is run will result in different cuckoo solutions. Finally, all the solutions across the nodes are compared based on their cost and the best solution is selected and presented as the final solution.

Algorithm: Discrete Cuckoo Search Optimization

Input: Distance Matrix

Output: Optimized path and cost

1: numNest = sizeof(InputMatrix)

2: Pa = 0.2 * numNest

3: MaxGen = 50

4: Generate initial population with randomly chosen city and store in nests.

5: **while** (MaxGen)

6: Assign random index i from nests to Cnest

7: Calculate Levy Flight

8: **if** Levy Flight>2 :

Perform Double Bridge for Cnest with random values

9: **else**:

Perform 2-Opt-Move for Cnest with random values

end if

10: Get random index j from nests

11: **if** cost(nest_j) > cost(Cnest)

Replace j by the new solution;

end if

12: A fraction(Pa) of the worse nests are abandoned.

13: Sort the nests based on cost

end while

14: nest[0] is the solution

V. IMPLEMENTATION DETAILS AND RESULT ANALYSIS

In this proposed work, anew approach is proposed tosolve the TSP optimization using multiple nodes and performing the computation in paralleli. By parallelizing we reduce the computational load on each of the individual machines thus improving performance of the algorithm. We have achieved communication between the nodes using Message Passing Interface (MPI, specifically mpi4py) as it is highly suitable for distributed memory models and supports all high performance computing platforms. Parallelizing is done by splitting the work among different nodes of the cluster.

Beowulf cluster was established for parallelizing. Beowulf cluster is a muti-computer architecture which uses identical computers connected via a network and libraries, which produces a high performance parallel cluster thus allowing for faster computation. It supports a Non uniform memory access (NUMA) architecture. The cluster follows a client-server architecture. Beowulf clusters often use cheap commodity grade computers, resulting in an inexpensive supercomputer.

First, a sequential algorithm was implemented in python, and it was then converted into a parallelized using MPI libraries designed for python (mpi4py) and then was executed on the Beowulf cluster, with the server node running the driver program. The experimental results are efficacious and are presented to demonstrate the benefits of parallelizing the discrete cuckoo search algorithm for the optimization of the travelling salesman problem.

VI. RESULTS AND DISCUSSION

Choosing the best parallel programming paradigm is extremely crucial to parallelizing an algorithm. There are several libraries which enable parallel computation, few of them being MPI, OpenMP and Parallel Virtual Machine (PVM). In this case, MPI is chosen as the paradigm of choice due to the nature of our implementation and hardware and memory limitations. MPI acts as the de facto standard for message passing libraries and is used to implement algorithms in parallel. MPI allows parallelism both within a node by spawning processes or across nodes which is facilitated by message passing. This advantage allows parallelism over distributed memories where OpenMP which employs thread level parallelism fails, thus making MPI more suitable for our needs. OpenMP also lacks the reliable error handling capabilities, scalability. Synchronization between a subset of threads is not allowed which further limits is use.Similar to OpenMP, CUDA alsouses threads and is unsuitable for distributed environmentsand often data transfers in CUDA cause performancebottlenecks. CUDA also has extremely low portability andis mostly suitable for SIMD architecture and is not suitablefor many algorithms which greatly limit it use. On thecontrary, MPI is found to be more suitable for a larger rangeof problems and provides the user more control andprogrammability in terms of distribution and processsynchronization.

Beowulf cluster was established on Oracle VM VirtualBox using Python3 under 64-bit Ubuntu 18.04.1 LTS, IntelRCoreTM i7-7500U 2.70GHz CPU and 2GB of RAM. Secured Shell(SSH) OpenSSH_7.6p1, OpenSSL 1.0.2n is used. Network File System version Server nfs v4 is used in master node and Client nfs v4 are used in slave nodes.

MPI is run on64 bit Fedora OperatingSystem.Experiments are conducted on a system with IntelR XeonR CPU E3- 1225 v3 @ 3.20GHz and 15 GB of RAM.

The proposed work of basic/standard and improved

Published By:
Blue Eyes Intelligence Engineering &
Sciences Publication

Retrieval Number H6711068819 /19©BEIESP

DCS algorithms has been implemented using Python3 with mpi4py-1.3.1. Mpiexec (OpenRTE) 2.1.1 version is used.

A 1000 x 1000 random TSP dataset was generated using a python code and stored in a file. That dataset had values ranging from 1 to 100 as distances i.e. dataset with 1000 towns with costs between 0 and 100 units in real time.

The results of the comparison have shown that Discrete CS outperforms the sequential method for solving TSP when run parallelly using MPI in a single system with multi-core. But time taken to execute on multi-node Beowulf cluster is higher due to communication overhead.

Time taken for execution for a dataset of 100x100 :

Time taken when executed with MPI with Beowulf: 2.049s

Time taken when executed with justMPI: 0.067s

Time taken when executed sequentially:0.056s

Executing parallelly is not faster than executing sequentially for a 100x100 dataset. Parallelizing for a small dataset is not efficient because of overhead and delays.

Communication overhead is very high for Beowulf, so it takes more time to execute than sequential for small datasets such as these, but not for real-time big data.

Time taken for execution for dataset of size 1000x1000:

Time taken when executed parallelly on MPI: 0.114s

Time taken when executed sequentially: 0.197s

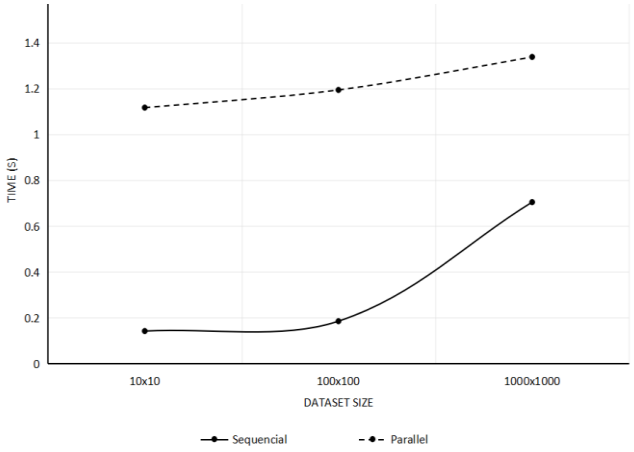
Therefore, executing parallelly is 42.13% faster than executing sequentially for a 1000x1000 dataset. For larger datasets, the speed up achieved by parallel execution when compared to sequential execution will be much higher.

The results are summarized in the tables below. The table shows the time taken when the three different dataset sizes are run sequentially and parallelly on Beowulf cluster with 3 nodes and on single node with 8 cores.

Table I. Time vs Dataset-size for Beowulf Cluster with three nodes

Dataset size	Sequential	Parallel
10 x 10	0.142s	1.116 s
100 x 100	0.185 s	1.193 s
1000 x 1000	0.704 s	1.337 s

Table I summarizes the result obtained in Beowulf cluster implementation. The time taken is denoted in seconds. Dataset size denotes the size of the input matrix given in the program.



(a) Parallel vs Sequential - Time vs Dataset-size Graph

The above graph is a time(s) vs dataset-size graph, representing sequential programming versus parallel programming time taken trends. It can be observed from the graph, that time varies must faster exponentially with respect to dataset size for sequential execution when compared to parallel execution. Therefore, although sequential programming is faster for small datasets (1000x1000), parallel programming will be faster for much larger datasets.

Table II. Time vs Dataset-size - Single node 8 cores

Dataset size	Sequential	Parallel
10 x 10	0.042 s	0.047 s
100 x 100	0.056 s	0.067 s
1000 x 1000	0.197 s	0.114 s

Table II summarizes the result obtained in this implementation. The time taken is denoted in seconds. Dataset size denotes the size of the input matrix given in the program.

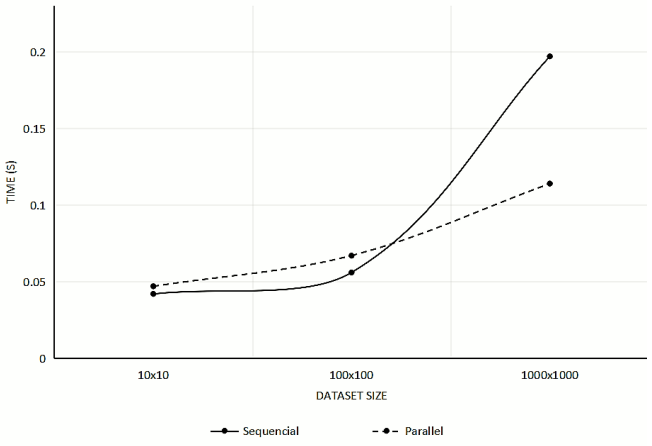


Fig. 2. Parallel vs Sequential - Time vs Dataset-size Graph

Fig. 2 is a time(s) vs dataset-size graph, representing sequential

programming versus parallel programming time taken trends. Although sequential programming is faster than parallel programming for smaller datasets, it is evident from the graph that there is definitely an upgrade in speed from sequential to parallel in the larger dataset i.e., 1000x1000

VII. CONCLUSION

This paper has tried to parallelize DCSO algorithm to improve the quality of the travelling salesman problem MPI. Discrete Cuckoo Search Optimization(DCSO) algorithm uses random walk with Levy flights. Hence, running it 'n' times provides us with n different solutions, out of which the best solution is taken. Since Real-time Travelling SalesmanProblems involves a much largerdataset where parallelism becomes essential.
The code is run sequentially 'n' times and parallelly in 'n' nodes and the time taken is noted. It is observed that although sequential programming may be faster for very small datasets, there is a considerable improvement in the time taken for larger datasets.

VIII. SCOPE FOR IMPROVEMENT

The existing system can be enhanced by improving node capacities or by reducing the communication overhead time, thus decrease the time taken for execution in a Beowulf cluster. Further parallelizing the DCSO algorithm, Increasing node capacities and decreasing the lag time in Network File System(NFS) can also lead to major headway in future expansions and implementations.

APPENDIX

Creating a Beowulf Cluster:
1. Create Virtual Machine(VM) with Ubuntu 18.04.1 LTS using Oracle VM Virtualbox.
2. Enable Bridged Networking between the noded
After installation, power off the virtual machines and for each node, do -
Settings -> Network -> Check the "Enable Network Adaptor" option -> Select "Bridged Adaptor" and select any one of the options that works-> OK
3.Install SSH on the all the nodes by using sudo apt-get install openssh-server openssh-client
4. Static IP Allocation
Assign a static IP address to nodes
ifconfig -> gives the network name, address, netmask, network address(X.X.0.0) and broadcast address
route -m -> gives the gateway address
cat /etc/resolv.conf -> gives the dns-nameservers address
d /etc/network
sudogedit interfaces
Add the following lines:
auto {insert-network-name}
iface {insert-network-name} inet static
address {insert-ip-address}
netmask {insert-network-mask}
network {insert-network-address}
broadcast {insert-broadcast-address}
gateway {insert-gateway-address}
dns-nameservers {insert-dns-nameservers-address}
5. Save the file and restart the system network
From home directory, open up hosts file in each node using -
cd /etc
sudogedit hosts
Add the following info in all the nodes -
{host1 IP address} {master-node-name}
{host2 IP address} {slave1-node-name}
{host3 IP address} {slave2-node-name}
6. Create Passwordless SSH from master to slave nodes
On the all the node, type ssh-keygen on the terminal. Just hit enter for all other options. This option will create a id_rsa.pub RSA public key for your node.
On the master node, type ssh-copy-id -i ~/.ssh/id_rsa.pub <first-node-name> on the terminal. It will prompt you for the password of the first-node, enter that. Repeat this step for all nodes. This will copy your RSA public key of your master node to the id_rsa.pub of your other nodes, enabling a passwordless SSH from master to all the slave node.
To check if this works, try ssh<node-name>. Try out for all nodes.
7. Setting up NFSon all the nodes
NFS - Network File System. Used to view, store and update files between local and remote systems. Here, we'll use it to provide a shared memory to the master and slave nodes.
On the master node, run the command sudo apt-get install nfs-kernel-server and install the NFS server. Create the folder that will be shared among the master and slave nodes using mkdir<foldername> . Move to etc folders and open up the exports file using cd /etc and sudogeditexports . Add the following at the end
-/home/<username>/<foldername><slave1>(rw,sync,no_root_squash,no_subtree_check)
<slave2>(rw,sync,no_root_squash,no_subtree_check)
Go back to home directory, run exportfs -a in the terminal followed by sudo service nfs-kernel-server restart
On the slave nodes, run the command sudo apt-get install nfs-common. Create a file with the same name as the one on the master node <foldername>. Run sudo mount -t nfs<master-node-name>:/home/<username>/<foldername> ~/<foldername>. To check if it's mounted or not, run df -h.
To permanently mount it, go to /etc folder and open up fstab file using cd /etc and sudogeditfstab. Add the following at the end -
<master-node-name>:/home/<username>/<foldername> /home/<username>/<foldername>
8. Install same version of mpi4py on all nodes of the cluster.
sudo apt-get install python-mpi4py
sudomkdir mpi4py
cd mpi4py
Download mpi4py-1.3.1 and move it to mpi4py directory.
tar xfz mpi4py-1.3.tar.gz
cd mpi4py-1.3/demo
Sample codes will be available in the demo directory.
9. Create a text file called hostfile.txt in the shared directory. In that directory, save either the nodes names (or their IP addresses) like mentioned below:
<master-node-name>
<slave-node-name-1>
<slave-node-name-2>
...
9. Running the code
To run the mpi code, save the executable object file of

Published By:
Blue Eyes Intelligence Engineering &
Sciences Publication
Retrieval Number H6711068819 /19©BEIESP

the code, in the common shared folder and run the following command in the terminal
chmod -c cc.py
mpirun -n 3 -hostsfilehostfile.txt ./cc.py

REFERENCES

1. Bianchi, Leonora; Marco Dorigo; Luca Maria Gambardella; Walter J. Gutjahr (2009). "A survey on metaheuristics for stochastic combinatorial optimization". Natural Computing: An International Journal. 8 (2): 239–287. doi:10.1007/s11047-008-9098-4
2. Blum, C.; Roli, A. (2003). "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". 35 (3). ACM Computing Surveys: 268–308.
3. Lawler, E. L. (1985). The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization (Repr. with corrections. ed.). John Wiley & sons. ISBN 978-0471904137.
4. Johnson, D. S.; McGeoch, L. A. (1997). "The Traveling Salesman Problem: A Case Study in Local Optimization" (PDF). In Aarts, E. H. L.; Lenstra, J. K. Local Search in Combinatorial Optimisation. London: John Wiley and Sons Ltd. pp. 215–310.
5. Gao, H.-C & Feng, B.-Q & Zhu, L. (2006). Reviews of the meta-heuristic algorithms for TSP. 21. 241-247+252.
6. Ji, J & Huang, Z & Liu, C & Dai, Q. (2010). An ant colony algorithm based on multiple-grain representation for the traveling salesman problems. JisuanjiYanjiuyuFazhan/Computer Research and Development. 47. 434-444.
7. Choong, Shin Siang & Wong, Li-Pei & Peng Lim, Chee. (2019). An artificial bee colony algorithm with a Modified Choice Function for the traveling salesman problem. Swarm and Evolutionary Computation. 44. 622-635. 10.1016/j.swevo.2018.08.004.
8. X.-S. Yang; S. Deb (December 2009). Cuckoo search via Lévy flights. World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). IEEE Publications. pp. 210–214. arXiv:1003.1594v1.
9. R. N. Mantegna, Fast, accurate algorithm for numerical simulation of Lévy stable stochastic processes, Physical Review E, Vol.49, 4677-4683 (1994).
10. M. Gutowski, Lévy flights as an underlying mechanism for global optimization algorithms, ArXiv Mathematical Physics e-Prints, June, (2001).
11. Jati, G.K. &Manurung, Ruli&Suyanto, Suyanto. (2012). Discrete cuckoo search for traveling salesman problem. 993-997.
12. Ouaarab, Aziz &Ahiod, Belaïd& Yang, Xin-She. (2014). Discrete cuckoo search algorithm for the travelling salesman problem. Neural Computing and Applications. 24. 10.1007/s00521-013-1402-2.
13. N. Tzy-Luen, Y. T. Keat and R. Abdullah, "Parallel Cuckoo Search algorithm on OpenMP for traveling salesman problem," 2016 3rd International Conference on Computer and Information Sciences (ICCOINS), Kuala Lumpur, 2016, pp. 380-385.
14. Majumder, Arindam &Laha, Dipak. (2018). Cuckoo Search on Parallel Batch Processing Machines. 10.1007/978-981-10-6872-0_62.

AUTHORS PROFILE

Bhavana V is currently a Student of Computer Science at VIT, Chennai. Main focus of her research is Parallel Processing and Machine Learning. Her primary domains include High Performance Computing, Machine Learning and Distributed Computing

Varshini Ramesh is currently a Student of Computer Science at VIT, Chennai. She is a passionate researcher in High Performance Computing and Machine Learning. Her primary research domains are Parallel Processing, Data Visualization and Machine Learning

M Sivagami is an Associate Professor in the School of Computing Sciences and Engineering, VIT, Chennai, India. She has published various technical papers in International Journals/Proceedings of International Conferences/Edited book chapters. She is a reviewer of some international journals. Her current research interests Data Clustering Techniques and Applications, Content Based Learning, Knowledge Representation and Reasoning, Information Retrieval, Big Data Analytics.