MapReduce-based Optimization of Overlay Networks using Particle Swarm Optimization

Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, USA
simone.ludwig@ndsu.edu

ABSTRACT

An overlay network is a virtual network that is built on top of the real network such as the Internet. Cloud computing, peer-to-peer networks, and client-server applications are examples of overlay networks since their nodes run on top of the Internet. The major needs of overlay networks are content distribution and caching, file sharing, improved routing, multicast and streaming, ordered message delivery, and enhanced security and privacy. The focus of this paper is the optimization of overlay networks using a Particle Swarm Optimization (PSO) approach. However, since the ever growing need for more infrastructure causes the number of network nodes to grow significantly, the parallelization of the PSO approach becomes a necessity. In this paper, the MapReduce concept, proposed by Google, is adopted for the PSO approach in order to be able to optimize large-scale networks. MapReduce is easy to implement since it is based on the divide and conquer method, and implementation frameworks such has Hadoop allow for scalability and fault tolerance. Experiments of the MapReduce based PSO algorithm are performed to investigate the solution quality and scalability of the approach.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

Keywords

Evolutionary computation, overlay network optimization

1. INTRODUCTION

Overlay networks are networks that are built on top of the physical network and rely on basic networking functions such as routing and forwarding of the physical network. The nodes of an overlay network are connected via logical links, and these can span over multiple physical links. The nodes of the overlay network are expected to meet the following requirements [1]:

- Provide the infrastructure to support the execution of one or more distributed application(s).
- Support high-level routing and forwarding tasks necessary in a network. The overlay network should provide data-forwarding capabilities that are different from the ones that are part of the Internet.
- Deployment should be across the Internet to allow third parties to participate in the organization and operation of overlay networks.

The main properties of overlay networks are adaptability and robustness. These two features are the major driving force behind the research of overlay networks. The aim of this paper is the optimization of an overlay network in terms of cost, performance, and reliability. In particular, the main focus of the optimization is the application of data mirroring.

Optimization of overlay networks has received significant attention. For example, the design of Service Overlay Networks (SON) is optimized by keeping reliability constraints in mind [2]. Two subfunctions, maximum profit and minimum cost are optimized using Lagrange multipliers.

Considering the scalability of an overlay network configuration, exact techniques are only applicable to solve very small sizes of overlay networks. Among a broad set of modern heuristic and metaheuristic methods for optimization, nature-inspired methods have emerged as promising techniques. For example, network design problems are solved by nature-inspired techniques since they are able to compute approximate solutions within a reasonable amount of time [3].

An Ant Colony Optimization approach is applied to the AntNet routing protocol [4]. Agents are used to concurrently explore the network and exchange collected information in the same way as ants explore the environment. It is the ants indirect communication capability depositing pheromone that is being exploited for the routing optimization.

A multi-objective evolutionary algorithm called NSGA-II was used in [5] to optimize a multicast overlay network. The aim is to minimize the total end to end delay of a multicast tree, and to maximize the link utilization.

A multi-swarm approach for neighbor selection in peer-to-peer overlay networks is described in [6]. A multi-swarm interactive pattern is introduced to match the dynamic nature of P2P networks.

The reconfiguration of the topology and link capacities of an operational network using genetic algorithms is de-

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
Copyright 2014 ACM 978-1-4503-2662-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2576768.2598269.

scribed in [7]. The reconfiguration is done in order to adapt to changes of its operational conditions. For example, nodes and links might become unavailable, the traffic patterns might change, and the quality of service requirement and priorities of different users and applications might change suddenly.

Another genetic algorithm approach for the reconfiguration in autonomic computing systems was proposed in [8]. The genetic algorithm approach, called Plato, uses evolutionary processes to automate the decision-making process of an autonomic system. Plato dynamically evolves target reconfigurations at run time, and at the same time, balances the trade-offs between functional and non-functional requirements to changes in environmental requirements and conditions. In particular, Plato is designed for the reconfiguration of a collection of remote data mirrors for diffusing data, and minimizing operational costs, and maximizing data reliability and network performance.

This paper closely follows the Plato approach, however, instead of using a genetic algorithm approach, a Particle Swarm Optimization (PSO) approach is implemented and evaluated. In an earlier study [9, 10], we showed that PSO outperforms the Genetic algorithm-based approach. However, since the scalability of the overlay network optimization is limited by the computational capability of the computer the algorithm is running on, a new parallelization approach is needed in order to enable the optimization of very large overlay networks.

The growth of the internet has challenged researchers to develop new ideas to deal with the ever increasing amount of computing infrastructure. The number of networking nodes is steadily increasing and therefore, more scalable algorithms need to be developed. Parallelization of the optimization algorithms is such option. Classic parallel applications that were developed in the past either used message passing runtimes such as MPI [11] or PVM (Parallel Virtual Machines) [12]. Both concepts utilize a rich set of communication and synchronization constructs, however, these need to be explicitly programmed. In order to make the development of parallel applications easier, Google introduced a programming paradigm called MapReduce that uses the *map* and *reduce* primitives that are present in functional programming languages. The MapReduce implementation enables large computations to be divided into several independent map functions. MapReduce provides fault tolerance since it has a mechanism that automatically re-executes *map* or *reduce* tasks that have failed.

The MapReduce model works as follows. The input of the computation is a set of key-value pairs, and the output is a set of output key-value pairs. The algorithm to be parallelized needs to be expressed as *map* and *reduce* functions. The *map* function takes an input pair and returns a set of intermediate key-value pairs. The framework then groups all intermediate values associated with the same intermediate key and passes them to the *reduce* function. The *reduce* function uses the intermediate key and set of values for that key. These values are merged together to form a smaller set of values. The intermediate values are forwarded to the Reduce function via an iterator. More formally, the *map* and *reduce* functions have the following types:

$map(k_1, v_1) \rightarrow list(k_2, v_2)$
$reduce(k_2, list(v_2) \rightarrow list(v_3)$

The main focus of this paper is to adapt Google's MapReduce paradigm for our Particle Swarm Optimization algorithm applied to the optimization of overlay networks in order to be able to optimize large-scale networks by making use of parallel computing. This enables large-scale networks above sizes of 500 nodes and 124750 links to be optimized. Previous experiments revealed that it is possible to run the optimization on a single computer for a network size of 500, however, for a 550-node network the algorithm did not execute anymore.

The paper is outlined as follows. Section 2 describes related work in the area of optimization for which MapReduce was adopted. In Section 3, the MapReduce-based PSO overlay optimization approach is described. Section 4 describes the experimental setup, and in Section 5 the results obtained are outlined. Section V contains the conclusions of this study.

2. RELATED WORK

The MapReduce paradigm has found widespread use in many fields of study. For example, it has been applied in the areas of spatial data processing, information retrieval, machine learning, data mining, bioinformatics, etc.

Since our paper addresses an optimization problem we review related work in the area of optimization for which MapReduce has been adopted. For example, MapReduce has been applied to a differential evolution algorithm in [13]. The authors demonstrate how easily MapReduce can be applied by parallelizing the fitness evaluation of the algorithm, which is the most time-consuming portion of the algorithm. Two alternative approaches were investigated - population based and data based. The experimental results suggested that even though the population based approach is the better way to proceed due to simplicity reasons, however, the extra cost of I/O operations and system overhead significantly reduces the benefits of parallelism.

In [14], MapReduce is applied to a variety of learning algorithms including locally weighted linear regression, k-means, logistic regression, naive Bayes, support vector machine, independent component analysis, principal component analysis, gaussian discriminant analysis, EM, and backpropagation. The authors' experimental results show a near to linear speedup with increasing numbers of processors.

An asynchronous implementation of MapReduce was investigated in [15]. The paper investigates partial synchronizations for iterative MapReduce applications to overcome global synchronization overheads. The approach applies a locality-based partitioning method. Local computations with relatively frequent local synchronizations and with less frequent global synchronizations are executed as mapper tasks. The performance gains are demonstrated for PageRank, shortest path and k-means.

The following related work is with regards to MapReduce applied to nature-inspired algorithms. Since nature-inspired algorithms are population-based algorithms for which all individuals in the population are evaluated by a fitness function that can be executed in parallel, this makes nature-inspired algorithms perfect candidates for parallelization. [16] shows how genetic algorithms can be modeled with the MapReduce paradigm. The experiments conducted showed the convergence and scalability of the approach on the OneMax optimization [17] problem for up to 10^5 variables.

Another Genetic algorithm based MapReduce approach is presented in [18]. The authors argue since a genetic algorithm is iterative in nature, an extension to the MapReduce model is necessary using a hierarchical reduction phase in order to speed up the computation further.

A MapReduce Particle Swarm Optimization approach is implemented to solve benchmark functions in [19]. The paper describes how MapReduce is adopted without explicitly addressing the parallelization details such as communication and synchronization. Their approach is applied on a benchmark function and the authors note that the MapReduce concept is not appropriate for easy benchmark functions since the parallelization overhead is outweighing the speedup gain obtained by parallelizing the function evaluations.

An intrusion detection system based on a parallel particle swarm optimization clustering algorithm using the MapReduce methodology is proposed in [20]. Particle swarm optimization is used for the clustering task and it shows that the sensitivity problem of initial cluster centroids as well as premature convergence are avoided. The experimental results on a real intrusion data set show that the proposed intrusion detection system scales very well with increasing data set sizes. In addition, close to linear speedup is achieved by improving the intrusion detection and false alarm rates.

The focus of this paper is to investigate the scalability of a MapReduce implemented Particle Swarm Optimization approach that is applied to the optimization of overlay networks. In particular, the scalability is investigated in terms of increasing network sizes, increasing numbers of available computational nodes, and increasing numbers of particles (i.e., population).

3. MAPREDUCE-ENABLED OVERLAY NETWORK OPTIMIZATION USING PSO

3.1 Overlay Network Optimization Formulation

The optimization in this paper is based on the dynamic reconfiguration of a collection of remote data mirrors. Data copies of critical data, in remote data mirroring, is stored at one or more secondary site(s) that prevent the protected data from failures that may affect the primary copy [21]. Two design criteria for remote data mirroring are given; the choice of the type of network link connection to the mirrors, and the choice of the remote mirroring protocol. Each link in the network is associated by the cost, throughput, latency and loss rate in order to determine the overall remote mirror design performance [21]. Furthermore, there are two types of remote mirroring protocols that affect the network performance and data reliability, namely synchronous and asynchronous propagation protocols.

The optimization design criteria are the same as for Plato [8]. The main goal is the construction of an overlay network for the data to be distributed to all nodes within the following constraints: (1) Overlay network must remain connected at all times; (2) Overlay network should never succeed the allocated monetary budget; (3) The data should be distributed as efficiently as possible, meaning the amount of bandwidth consumed when diffusing data should be minimized.

The author of this paper has implemented three optimization algorithms to evaluate and compare their performances.

Table 1: Link Propagation Methods

Time interval	**Avg. data batch size in GB**
0	0
1 min	0.0436
5 min	0.2036
1 hr	2.091
4 hr	6.595
12 hr	15.12
24 hr	27.388

An implementation of Genetic Algorithms as used in the Plato implementation [8], an implementation of the network selection algorithm inspired by Artificial immune systems, and an implementation of a discrete Particle Swarm Optimization approach have been implemented [20]. Since the PSO-based implementation showed the best performance, it is used in this study for the parallelization using MapReduce. However, we first describe details about the algorithm as well as the problem description of the overlay network optimization task before moving on to the description of the MapReduce adoption.

The fitness function that is used as a measure is slightly modified compared to the Plato approach [8]. The differences are that normalization of the overall fitness value is done as to obtain a value between 0 and 1, as well as the sum of weights for each part of the fitness function is 1.

The fitness function consists of three parts (as in Plato); the first part evaluates the overlay network in terms of cost, the second in terms of performance, and the third part evaluates the reliability of the overlay network. The overall fitness function (Equation 1) is the weighted average of all three fitness portions. Please note that the sum of the weights needs to sum up to 1 (Equation 2).

$$F_{overall} = w_1 * F_{cost} + w_2 * F_{perf} + w_3 * F_{rel} \tag{1}$$

$$\sum_{i=1}^{3} w_i = 1 \tag{2}$$

Looking at the different fitness sub-functions, the fitness sub-function for cost is given as:

$$F_{cost} = 1 - \frac{cost}{budget} \tag{3}$$

where *cost* is the sum of operational expenses of all active links, and *budget* is a user supplied value on the maximum amount of money for an operating overlay network. The sub-function for the performance consists of two parts: latency and bandwidth as given below:

$$\begin{aligned} F_{perf} &= 0.5 * (1 - \frac{latency_{avg}}{latency_{wc}}) \\ &+ 0.5 * (\frac{bandwidth_{sys} - bandwidth_{eff}}{bandwidth_{sys}} + bound) \end{aligned} \tag{4}$$

where $latency_{avg}$ is the average latency over all active links, and $latency_{wc}$ is the largest latency value measured over all links in the underlying network; $bandwidth_{sys}$ is the total available bandwidth across the overlay network given the active links, and $bandwidth_{eff}$ is the total effective bandwidth across the overall network after data has been coalesced, and *bound* is a limit on the best value that can be achieved throughout the network.

The last fitness sub-function measures the overlay network in terms of reliability consisting of two parts as given below:

$$F_{rel} = 0.5 * (\frac{links_{used}}{links_{max}}) + 0.5 * (1 - \frac{dataloss_{pot}}{dataloss_{max}}) \quad (5)$$

where $links_{used}$ is the number of active links, and $links_{max}$ is the maximum number of possible links given the network structure; and $dataloss_{pot}$ is the total amount of data that could be lost during write coalescing using the propagation methods as given in Table 1, and $dataloss_{wc}$ is the amount of data that could be lost during write coalescing using the propagation method with the largest time window.

3.2 Particle Swarm Optimization Implementation

Particle Swarm Optimization (PSO) as introduced in [22], is a swarm based global optimization algorithm. The algorithm is inspired by the behavior of bird swarms searching for an optimal food source. There are two standard kinds of PSO variants: global best PSO and local best PSO. Since we are using global best PSO, the movement of a single particle is influenced by its last movement, its knowledge, and the swarm's knowledge. PSO's basic equations are the following:

$$x_i(t+1) = x_i(t) + v_{ij}(t+1) \quad (6)$$

$$v_{ij}(t+1) = w(t)v_{ij}(t) + c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t))) + c_2 r_{2j}(t)(\hat{y}_j(t) - x_{ij}(t))) \quad (7)$$

where x represents a particle, i denotes the particle's number, j the dimension, t a point in time, and v is the particle's velocity. y_{ij} is the best location the particle ever visited (the particle's knowledge), and $\hat{y}_j$ is the best location any particle in the swarm ever visited (the swarm's knowledge). w is the inertia weight and used to weigh the last velocity, c_1 is a variable to weigh the particle's knowledge, and c_2 is a variable to weigh the swarm's knowledge. r_1 and r_2 are uniformly distributed random numbers between zero and one. The pseudo code of the global best PSO is given in Algorithm 1 [23].

Since the overlay network optimization is a discrete problem, and PSO was designed for continuous problems, several operations have to be defined. This discrete implementation follows in part the implementation for solving the traveling salesman problem as described in [24]. First, a swarm of particles is initialized. A single particle represents one overlay network, i.e., every particle's position in the search space has to correspond to a possible overlay network. Velocities are implemented as lists of changes that can be applied to a particle (its vector) and that moves the particle to a new position (a new overlay network). Changes are exchanges of values of the overlay network. Furthermore, the difference between two matches (particles), multiplication of a velocity with a real number, and the addition of velocities have to be defined. The difference is implemented as a function of particles which returns the velocity containing all changes that

Algorithm 1 PSO Algorithm

```
Initialize an $n_x$-dimensional swarm S
repeat
    for each particle $i = 1, ..., S.n_s$ do
        if $f(S.x_i) < f(S.y_i)$ then
            $S.y_i = S.x_i$
        end if
        if $f(S.y_i) < f(S.\hat{y})$ then
            $S.\hat{y} = S.y_i$
        end if
    end for
    for each particle $i = 1, ..., S.n_s$ do
        updateVelocity()
        updatePosition()
    end for
until stopping criterion is satisfied
```

Table 2: PSO parameters

Parameter	Value
Number of particles	100
Number of iterations	500
Inertia weight	0.001
Weight of local knowledge	0.5
Weight of global knowledge	0.5

have to be applied to move from one particle's position to another in the search space. Multiplication randomly deletes single changes from the velocity vector if the real number to be multiplied is smaller than one. If the real number is one, no changes are applied. For a real number larger than one, random changes are added to the velocity vector. Table 2 shows the specific parameters chosen for the PSO implementation as identified by preliminary experiments.

3.3 MapReduce-based PSO Implementation

For the experiments in this paper, we are making use of Apache Hadoop [25], which is the commonly used MapReduce implementation. It is an open source framework that supports data-intensive distributed applications and enables applications to work with thousands of computational independent computers and petabytes of data. Hadoop Distributed File System (HDFS - storage component) and MapReduce (processing component) are the main core components of Apache Hadoop. HDFS allows high-throughput access to the data while maintaining fault tolerance by creating multiple replicas.

As mentioned before, a MapReduce implementation consists of three parts, the main, *map* and *reduce* functions. The main function first reads the overlay network parameters that are being used by the fitness function and writes it to the HDFS in order to be used by the *reduce* function. It then initializes the swarm by randomly initializing particles. Then the iteration loop of the algorithm starts by going through each particle and the Hadoop framework calls and executes the *map* and *reduce* functions, respectively. The pseudo code in Algorithm 2 shows the main function.

MapReduce's main operations are the *map* function and the *reduce* function. As shown in Algorithm 3, the *map* function reads the global best network from the HDFS. Given the global best network, the PSO update portion is applied in order to generate a modified network. This network, which

Algorithm 2 Main Function

```
procedure MAIN($maxIteration$)
    $network$ = readsOverlayNetworkParametersFromFile()
    writesOverlayNetworkParametersToHDFS($network$)
    $particles[]$ = initializeSwarm()
    $iteration = 0$
    while $iteration \leq maxIteration$ do
        for all $particles$ do
            hadoopCallsAndExecutesMapFunction()
            hadoopCallsAndExecutesReduceFunction()
        end for
    end while
end procedure
```

is the particle including the particle ID, is then emitted and can be used by the *reduce* function.

Algorithm 3 Map Function

```
procedure MAP($Key$: $ParticleID$, $Value$: $Particle$)
    $particleID$=$Key$
    $particle$=$Value$
    readGlobalBestFromHDFS()
    applyDiscretePSOUpdates($particle$)
    emit($particleID$, $particle$)
end procedure
```

The *reduce* function reads the global best network from the HDFS first. Then, the network parameters are read from the HDFS, and the particle's fitness is calculated. Afterwards, the fitness value is then compared to the global best fitness value. If the particle's fitness is greater than the current best, the particle is written to the HDFS, i.e., the network is written to the HDFS. At the end, the particle ID and the particle are emitted to be used in the next iteration.

Algorithm 4 Reduce Function

```
procedure REDUCE($Key$: $ParticleID$, $ValueList$: $Particle$)
    $particleID$=$Key$
    $particle$=$Value$
    $globalBestFitness$ = readGlobalBestFromHDFS()
    $network$ = readNetworkParametersFromHDFS()
    $fitness$ = calculateFitness($particle$,$network$)
    if $fitness \leq globalBestFitness$ then
        $globalBestFitness = fitness$
        writeGlobalBestToHDFS($particle$)
    end if
    emit($particleID$, $particle$)
end procedure
```

4. EXPERIMENTAL SETTINGS

In this section, we describe the experimental setup of the MapReduce based PSO overlay optimization algorithm. First, the computing environment used is described, followed by a description of the different networks used for the experiments.

4.1 Computing Environment

The experiments were conducted on the Longhorn Hadoop cluster hosted by the Texas Advanced Computing Center (TACC)[1]. The TACC cluster consists of 48 nodes containing 48GB of RAM, 8 Intel Nehalem cores (2.5GHz each) which results in 384 compute cores with 2.304 TB of aggregated memory. Hadoop version 0.21 was used for the MapReduce framework, and Java runtime 1.7 for the system implementation. Furthermore, in order to provide a constant level of parallelization, the number of mapper and reducer tasks were set to the maximum, which are 8 per node since each node contains 8 cores.

[1]https://portal.longhorn.tacc.utexas.edu/

Table 3: Different networks used for scaling experiments

Number of nodes	Number of links	File size
100	4,950	187KB
200	19,900	775KB
300	44,850	1.8MB
400	79,800	3.1MB
500	124,750	4.9MB
600	179,700	7.1MB
700	244,650	9.7MB
800	319,600	12.7MB
900	404,550	16.1MB
1,000	499,500	19.1MB

4.2 Network Sizes

Table 3 shows the different network characteristics used for the scaling experiments. The number of nodes, number of links, and the file size are listed.

5. RESULTS

This section shows the results obtained from the experiments. Three different experiments were performed, the first investigated the fitness and running time when the number of iterations are scaled, the second looked at the fitness and running time when the number of network nodes are scaled, and the third investigated the running time and speedup when different numbers of computing nodes are used. All measurement points shown in the graphs are average values of 25 independent runs.

5.1 Scaling of number of iterations

First of all, experiments were run to capture the solution quality and the running time of the algorithm per iterations, as shown in Figures 1 and 2, respectively. A 300-node network was used, and the particle size was set to 100 using the fitness equation with equal weights (as shown in Equation 1). Two computation nodes of the TACC environment were used for these experiments. Figure 1 shows that the optimization converges after around 270 iterations to the optimum value of 8.33.

As for the running time, as expected we can see that each iteration takes roughly the same amount of time resulting in a linear trend (with only slight variations) for increasing iterations.

5.2 Scaling of number of network nodes

The next set of experiments investigated again the solution quality and running time, however, this time not the number of iterations were increased but the network sizes were increased. The number of iterations was fixed to 500, and network sizes between 100 to 1,000 with increments of 100 were used. The number of computational nodes used was 2 for this experiment. In terms of fitness, the values up to network sizes of 400 produce the optimal fitness of

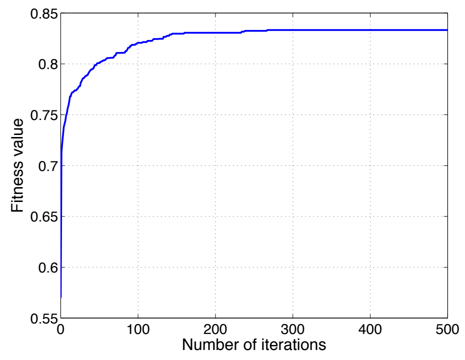


Figure 1: Fitness value versus number of iterations

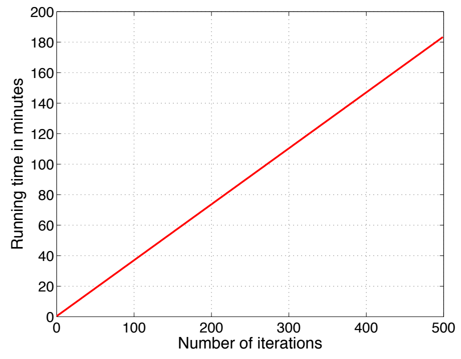


Figure 2: Running time versus number of iterations

0.833, however, for larger network sizes the fitness reduces as shown in Figure 3. In order to achieve the optimal fitness value the number of iterations would need to be increased for networks of size 500 and higher.

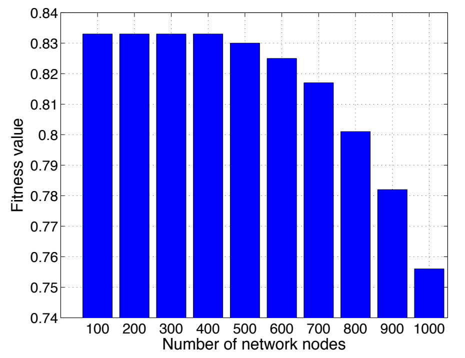


Figure 3: Fitness value versus number of nodes

In terms of running time, we can see the effect of increasing numbers of network sizes in Figure 4. An exponential trend is observed measuring 5.3 minutes and 195.2 minutes for a network of 100 nodes and 1,000 nodes, respectively.

5.3 Scaling of number of computational nodes

Since speedup is the measure to evaluate parallel algorithms, we used the following speedup equation:

$$Speedup = \frac{T_2}{T_n} \tag{8}$$

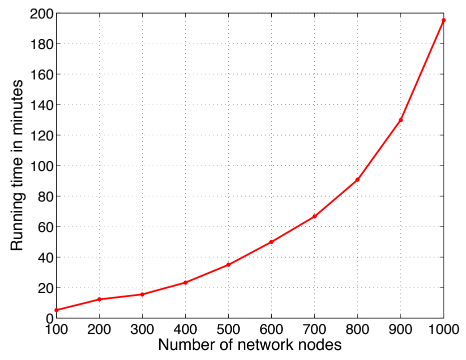


Figure 4: Running time versus number of nodes

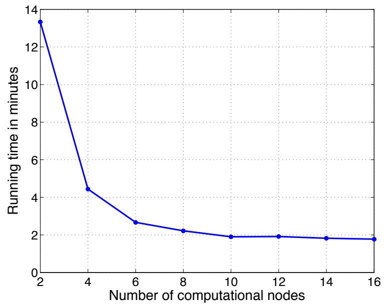
where T_2 is the running time using 2 nodes, and T_n is the running time using n nodes, where n is a multiple of 2.

For these sets of experiments, we increased the number of computational nodes by 2 starting from 2 compute nodes up to 16 compute nodes. The running time is measured and the speedup is calculated by Equation 8. Experiments were run for a 500-node network using 200, 400, and 600 particles, respectively. The number of iterations were kept constant with 500. The black line in all speedup figures shows the ideal speedup. The closer the measured speedup gets to the line, the better the utilization of the parallel computing nodes is. Figure 5 shows the results. As can be seen from the figure, the speedup is worse for fewer numbers of particles achieving a speedup of around 6.5 for 200 particles, a speedup of around 10 for 400 particles, and a speedup of around 12 for 600 particles. This demonstrates that the utilization of the Hadoop framework is better when larger numbers of particles are used. This implies that the fitness evaluations of the particles that are partitioned onto the different compute nodes is more efficient when more computation is necessary. Therefore, the control and communication overhead of the Hadoop framework does not have as large as an effect for larger numbers of particles.

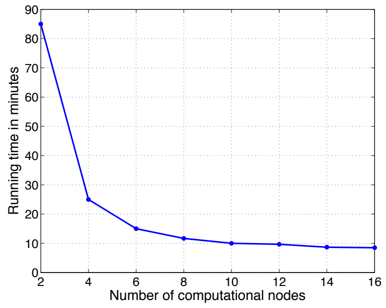
Figure 6 shows similar behavior of running time and speedup (please note that the running times are this time in hours and not in minutes as in Figure 5). However, a 1,000-node network was used and initial experiments with the same numbers of particles as for the 500-node network did produce poor results. Therefore, the particle sizes of 400, 800, and 1,200 were used for this set of experiments. The interesting observation, as seen in the figure, is that the speedup in all cases turn out to be better as compared to the 500-node network. Again, this can be explained with a better utilization of the Hadoop framework resulting in higher speedup gains when larger particle sizes are used. A speedup of around 10 is achieved with 400 particles, a speedup of almost 12 is obtained when 800 particles are used, and the speedup for 1,200 particles is close to 14.

6. CONCLUSION AND FUTURE WORK

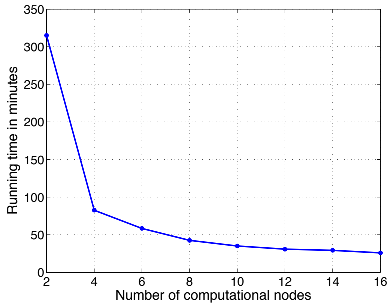
The focus of this paper was the parallelization of a Particle Swarm Optimization (PSO) approach for the optimization of overlay networks. For the parallelization, the MapReduce paradigm was exploited. A MapReduce based PSO overlay network optimization algorithm was implemented and extensive experiments were conducted to evaluate the solution



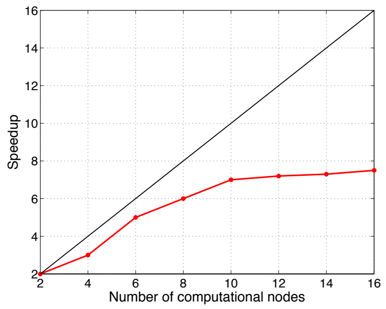
(a) Running time for 200 particles



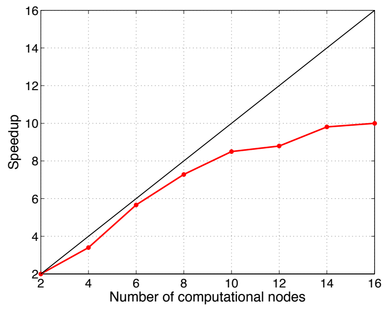
(b) Running time for 400 particles



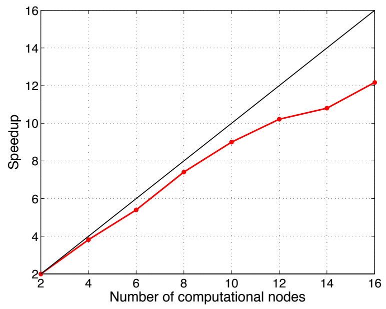
(c) Running time for 600 particles



(d) Speedup for 200 particles



(e) Speedup for 400 particles



(f) Speedup for 600 particles

Figure 5: Running time and speedup results for network of 500 nodes with 200, 400, and 600 particles

quality, running time and speedup for increasing numbers of network nodes. Networks consisting of 100 up to 1,000 nodes were investigated. The results revealed that the utilization of the Hadoop framework increased with increasing network sizes as well as with increasing numbers of particles used. A general recommendation can be made that the larger the network to be optimized, the larger the number of particles used within the PSO algorithm has to be. Reflecting on the particular speedup measurements conducted, good speedups can be achieved using 10 or 12 computational nodes.

Further work includes the investigation of how long each function per iteration runs in order to find out which of the three portions (main, map, or reduce function) of the code runs the longest. In addition, larger sizes of networks over 1,000 nodes could be optimized, however, running times of several days are expected.

Acknowledgments

The author acknowledges the support of the NDSU Advance FORWARD program sponsored by NSF HRD-0811239 and ND EPSCoR through NSF grant EPS-0814442. This work would have not been possible without the Extreme Science and Engineering Discovery Environment (XSEDE) infrastructure, which is supported by the National Science Foundation under grant number OCI-1053575.

7. REFERENCES

[1] S. Tarkoma, "Overlay Networks: Toward Information Networking", CRC Press, Auerbach Publications, ISBN: 978-1-4398-1371-3, 2010.

[2] N. Lam, Z. Dziong, L. G. Mason, "Service Overlay Network Design with Reliability Constraints", Proceedings of IEEE 7th International Workshop on the Design of Reliable Communication Networks, Washington, D.C., USA, 2009.

[3] T. Baeck, D. Fogel, and Z. Michalewicz, "Handbook of Evolutionary Computation", IOP Publ. Ltd., Bristol, UK, 1997.

[4] G. D. Caro, M. Dorigo, "AntNet: distributed stigmergetic control for communications networks", J. Artif. Int. Res. 9, 1 (December 1998), 317-365, 1998.

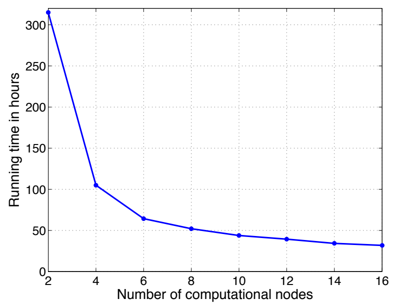
[5] J. Montoya, Y. Donoso, E. Montoya, D. Echeverri, "Multiobjective model for multicast overlay networks over IP/MPLS using MOEA", Proceedings of International Conference on Optical Network Design and Modeling, 1-6, 2008.

[6] A. Abraham, H. Liu, Y. Badr, C. Grosan, "A multi-swarm approach for neighbor selection in peer-to-peer networks", Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology (CSTST '08), ACM, New York, NY, USA, 178-184, 2008.

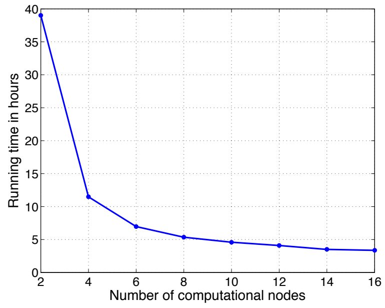
[7] D. Montana, T. Hussain, T. Saxen, "Adaptive reconfiguration of data networks using genetic algorithms", Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1141-1149, San Francisco, CA, USA, 2002.

[8] A. J. Ramirez, D. B. Knoester, B. H. C. Cheng, P. K. McKinley, "Plato: A Genetic Algorithm Approach to Run-Time Reconfiguration in Autonomic Computing Systems", Journal of Cluster Computing, 2010.

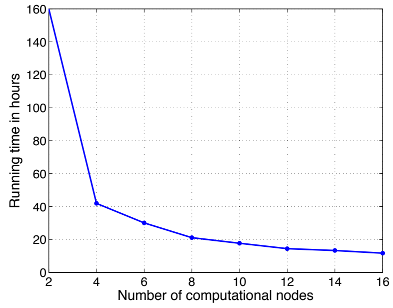
[9] S. A. Ludwig, "Nature-Inspired Reconfiguration of Overlay Networks", Proceedings of Third World Congress on Nature and Biologically Inspired Computing (NaBIC), Salamanca, Spain, 2011.



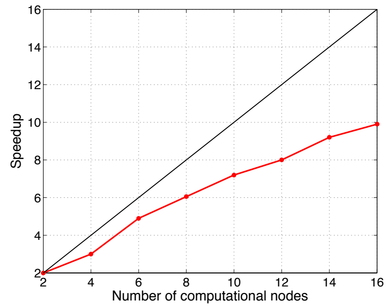
(a) Running time for 400 particles



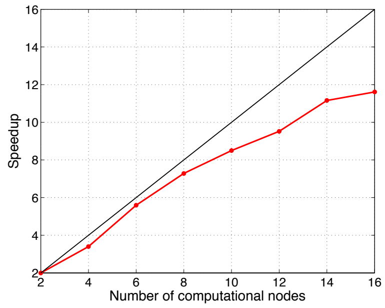
(b) Running time for 800 particles



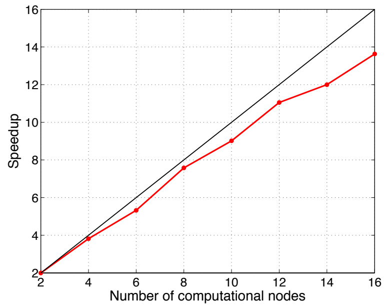
(c) Running time for 1,200 particles



(d) Speedup for 400 particles



(e) Speedup for 800 particles



(f) Speedup for 1,200 particles

Figure 6: Running time and speedup results for network of 1,000 nodes with 400, 800, and 1,200 particles

[10] S. A. Ludwig, "Scalability Analysis: Reconfiguration of Overlay Networks Using Nature-Inspired Algorithms", in Advances in Intelligent Modelling and Simulation: Artificial Intelligence-based Models and Techniques in Scalable Computing, Studies in Computational Intelligence series, vol. 422, pp. 137-154, Springer, 2012.

[11] MPI (Message Passing Interface), 2014, `http://www-unix.mcs.anl.gov/mpi/`.

[12] PVM (Parallel Virtual Machine), 2014, `http://www.csm.ornl.gov/pvm/`.

[13] C. Zhou, "Fast parallelization of differential evolution algorithm using MapReduce", Proceedings of the 12th annual conference on Genetic and evolutionary computation, 2010.

[14] C. T. Chu, S. K. Kim, Y. A. Lin, et al., "Map-Reduce for Machine Learning on Multicore", In NIPS (2006), pp. 281-288 by edited by Bernhard Schoelkopf, John C. Platt, Thomas Hoffman.

[15] K. Kambatla, N. Rapolu, S. Jagannathan, and A. Grama, "Asynchronous Algorithms in MapReduce", Proceedings of the 2010 IEEE International Conference on Cluster Computing (CLUSTER '10), Washington, DC, USA, 245-254, 2010.

[16] A. Verma, X. Llora, D. E. Goldberg, and R. H. Campbell, "Scaling Genetic Algorithms Using MapReduce", Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications (ISDA '09), Washington, DC, USA, 2009.

[17] J. Schaffer and L. Shelman, "On Crossover as an Evolutionary Viable Strategy", In R. Belw and L. Booker, editors, Proceedisn of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann, 1991.

[18] C. Jin, C. Vecchiola, R. Buyya, "MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms", Proceedings of IEEE Fourth International Conference on eScience, 2008.

[19] A. W. McNabb, C. K. Monson, K. D. Seppi, "Parallel PSO using MapReduce", Proceedings of Congress of Evolutionary Computation (CEC), 2007.

[20] I. Aljarah and S. A. Ludwig, "MapReduce Intrusion Detection System based on a Particle Swarm Optimization Clustering Algorithm", Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, June 2013.

[21] K. Keeton, C. Santos, D. Beyer, J. Chase, J. Wilkes, "Designing for disasters", Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp. 59-62. Berkeley, CA, USA, 2004.

[22] J. Kennedy, R. Eberhart, "Particle swarm optimization", Proceedings of IEEE International Conference on Neural Networks, Perth, Western Australia, 1995.

[23] A. Engelbrecht, "Computational Intelligence - An Introduction", 2nd Edition, Wiley, 2007.

[24] M. Clerc, "Discrete particle swarm optimization - illustrated by the traveling salesman problem". New Optimization Techniques in Engineering, In Studies in Fuzziness and Soft Computing, Springer, 2004.

[25] Apache Software Foundation, Hadoop MapReduce, 2011. `http://hadoop.apache.org/mapreduce.`