Computer Communications 199 (2023) 177–185

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A generic parallel optimization framework for solving hard problems in optical networks☆

Longfei Li [a], Yongcheng Li [a], Sanjay K. Bose [b,1], Gangxiang Shen [a,*,2]

[a] Jiangsu New Optical Fiber Technology and Communication Network Engineering Research Center & Suzhou Key Laboratory of Advanced Optical Communication Network Technology & Institute for Broadband Research & Innovation (IBRI), School of Electronic and Information Engineering, Soochow University, Jiangsu Province 215006, PR China

[b] Plaksha University, Mohali, India

ARTICLE INFO

Keywords:
Optimization problems
Parallel optimization
Heuristic
Meta-heuristic
Mixed integer linear programming
Machine learning
Bin-packing problem
Optical networks

ABSTRACT

Many optimization problems in optical networks are either NP-hard or take a long time to solve. In this paper, we develop a generic parallel computing framework which either solves these problems completely or at least improves the quality of the solution. To verify the effectiveness of the proposed framework, we build a miniaturized parallel computing system and test our approach based on different heuristic and meta-heuristic algorithms for the bin-packing problems, mixed integer linear programming (MILP) models, and distributed machine learning. Experimental studies show that the proposed parallel computing framework based on the miniaturized parallel computing system is effective in so far as it either significantly reduces the computing time or improves the quality of the solution as compared to when only a single machine is used.

1. Introduction

There are many optimization problems that arise naturally in optical networks. Examples of these are the problems associated with routing and wavelength assignment [1], minimizing energy consumption [2], shared backup path protection (SBPP) network planning [3], and the adaptive forward error correction (FEC) code assignment [4]. Different approaches have been proposed to solve these problems, including mixed integer linear programming (MILP), heuristic or meta-heuristic approaches, as well as more advanced approaches based on machine learning. However, due to the NP-hardness/completeness of these problems, it would often take a long time for these approaches to efficiently find an optimal solution even when such a solution actually exists. This is especially true if one attempts this on a single machine, when the size of a problem is large. To improve the quality of the solution provided by these approaches, a straightforward way is to increase the computing resources being used, such as by using parallel computing with multiple machines, so as to jointly optimize the problem. Although it may be easy to set up a hardware system with multiple machines, the mechanism for parallel optimization would still be challenging as one would need to decompose the original optimization problem into multiple sub-problems or processes which can be guaranteed to be independent of each other.

To improve computing efficiency and the quality of solutions, we develop a generic decomposition mechanism for different optimization problems. This can decompose an optimization problem into multiple sub-problems or processes, each of which can then be independently solved in parallel on different machines. To verify the effectiveness of the proposed mechanism, we design a Hadoop-based parallel computing system, which was initially prototyped using multiple desktops and eventually evolved to an integrated miniaturized system. By implementing the parallel computing mechanisms for various optimization problems and approaches, we show that the proposed decomposition mechanism and the Hadoop-based system developed can significantly reduce the optimization time with at least the same solution quality compared to one without such decomposition, i.e., solving problems on a single machine. Moreover, though this study has only considered four typical optimization approaches, the proposed framework is generic enough to solve other optimization problems that can be naturally decomposed into multiple independent sub-problems. This would therefore be an efficient way to handle the computational difficulties faced by many optimization problems. The main contributions of this paper are as follows:

☆ This work was supported in part by the National Key R&D Program China under Grant 2018YFB1801701 and the National Nature Science Foundation of China under Grant 62171306.

* Corresponding author.
E-mail addresses: skbose@gmail.com (S.K. Bose), shengx@suda.edu.cn (G. Shen).
[1] Senior Member, IEEE.
[2] Senior Member, IEEE/Fellow, OPTICA.

https://doi.org/10.1016/j.comcom.2022.12.023
Received 23 February 2022; Received in revised form 20 July 2022; Accepted 22 December 2022
Available online 24 December 2022

First, we develop a generic mechanism for decomposing a complex optimization problem into multiple independent sub-problems/processes and then solve these sub-problems and processes in parallel to reduce computation difficulty. This mechanism can be applied to the optimization approaches proposed in different categories, such as heuristic, meta-heuristic, MILP, and machine learning.

Second, to demonstrate the effectiveness of the proposed decomposition mechanism, we design a miniaturized Hadoop-based parallel optimization system, which is capable of elastically expanding its computing capacity by interconnecting multiple system boxes as needed.

Third, experiments are conducted to solve optimization problems in various areas of optical networks. It is demonstrated that the proposed decomposition mechanism and the miniaturized parallel optimization system can effectively solve these optimization problems in much shorter times as compared to when a single machine is used.

The rest of this paper is organized as follows. In Section 2, we introduce related work in the literature. In Section 3, we describe the decomposition mechanism in the context of various optimization approaches including heuristic, meta-heuristic, MILP, and machine learning. In Section 4, we introduce the miniaturized parallel optimization system, elaborating on the system's evolution from multiple isolated desktops to an integrated compact box. We conduct experiments and verify the effectiveness of the proposed decomposition mechanism and the parallel optimization system in Section 5. This is done in the context of various optimization problems taken from the area of optical networks. We conclude the paper in Section 6.

2. Related work

Cloud computing takes advantage of large-scale computing clusters to analyze and process vast amounts of data while ensuring security, fault tolerance, load balancing, high concurrency, and scalability [5]. Major cloud computing service providers have developed distributed programming systems such as High-Performance Computing Cluster (HPCC), Apache Hadoop, and Apache Spark. HPCC is an open-source, data-intensive computing framework [6], based on a software architecture running on commodity computing clusters which can provide high-performance, data-parallel processing for big data applications. Similarly, Hadoop is also an open-source software framework for solving problems with large volumes of data and computational resources using a network of many computers [7]. However, Hadoop takes a processing model that frequently reads/writes data from/to the disk, which is not suitable for real-time applications. To support real-time applications, Spark is another open-source unified analytics engine for large-scale data processing, which uses the main memory to perform in-memory computation and therefore minimizes data transfer to and from the disk. Spark also provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. The codebase of Spark was first developed by AMPLab at the University of California, Berkeley, and donated to the Apache Software Foundation [8].

Considerable effort has been devoted to solving optimization problems using a cloud computing system. Based on a HPCC system, Hukkeri et al. [9] developed a novel framework to implement distributed image processing via OpenCV on the HPCC system. This showed a 30% decrease in computing time without sacrificing accuracy. Xu et al. [10] presented a massively scalable parallel K-Means method that leverages the distributed computing environment of the HPCC system and showed a significant scalability improvement for the parallel K-Means algorithm. Herrera et al. [11] implemented a Random Forest classifier based on the LexisNexis HPCC system. By taking full advantage of the HPCC system's big data processing capability, they enhanced data gathering from an inefficient *Pass them All and Filter* approach into an effective parallelized *Fetching on Demand* approach. Similar studies based on the HPCC system also include [12].

Based on the Hadoop system, Shen and Li developed a Hadoop-based cloud computing system to parallel solve bin-packing problems such as lightpath routing and wavelength assignment in an optical network [13,14]. The authors have also subsequently employed the Hadoop-based cloud computing system developed by them to solve problems in the areas of lightpath routing and wavelength assignment [1], energy minimization in IP over WDM network [2], shared backup path protection-based routing and spectrum assignment in an elastic optical network (EON) [3] and forwarding error correction code allocation in an EON [4]. Using a Hadoop platform, Shen et al. [15] employed a genetic algorithm to solve the traveling salesman problem to save computing time. Yildirim et al. [16] also ran the genetic algorithm on a Hadoop system to solve high-dimensional problems. Similar studies based on the Hadoop system also include [17,18].

Based on the Spark system, Matthew et al. [19] implemented a hybrid K-Means particle swarm optimization clustering algorithm and showed that Spark enables efficient scaling of resources to handle large and complex workloads. Chen et al. [20] proposed a parallel Ant Lion Optimizer algorithm to seek optimal parameter combinations on a Spark platform, and showed that compared with the traditional Ant Lion Optimizer algorithm, the Spark-based parallel scheme can effectively improve classification accuracy in the parameter-tuning process of Random Forest. Similar studies based on the Spark system also include [21–26].

Table 1 summarizes the related work on employing the above three types of computing systems to solve various optimization problems in communication networks, where different optimization approaches are used. These include heuristic, meta-heuristic, and machine learning approaches. Note that, for the integer linear programming (ILP) approach, we cannot find any related work leveraging the above three types of computing systems to tackle the optimization problems of communication networks.

Table 1
Reference on using cloud computing technique to solve optimization problems with different optimization approaches.

Optimization approach	Optimization problem	Reference
Heuristic	Routing and wavelength assignment	[13]
	SBPP-based network planning	[14]
	FEC assignment	[14]
	Energy minimization of IP over WDM	[14]
Meta-heuristic	Network resource allocation	[17]
	Network design	[18]
	Network deployment	[21]
	Network anomaly data detection	[22]
Machine learning	Network anomaly data detection	[23,24]
	Virtual topology design	[25]
	Transmission cost minimization	[26]

3. Decomposition mechanism for parallel optimization

In this section, we describe the mechanism of decomposing an optimization problem into multiple independent sub-problems/processes and then solving these sub-problems or running these processes in parallel to overcome the computing difficulties. We target four general categories of optimization approaches: heuristic, meta-heuristic, MILP, and machine learning. Note that, although the proposed decomposition mechanism is universal and potentially applicable for a wide range of optimization problems, in this paper we specifically focus on the optimization problems arising in the area of optical networks as the examples being considered.

3.1. Heuristic

In the area of optical networks, many optimization problems, e.g., lightpath routing and wavelength assignment (RWA) [1], IP over wavelength division multiplexing (WDM) traffic grooming [2] and others, belong to the category of the bin-packing problem, which is defined as

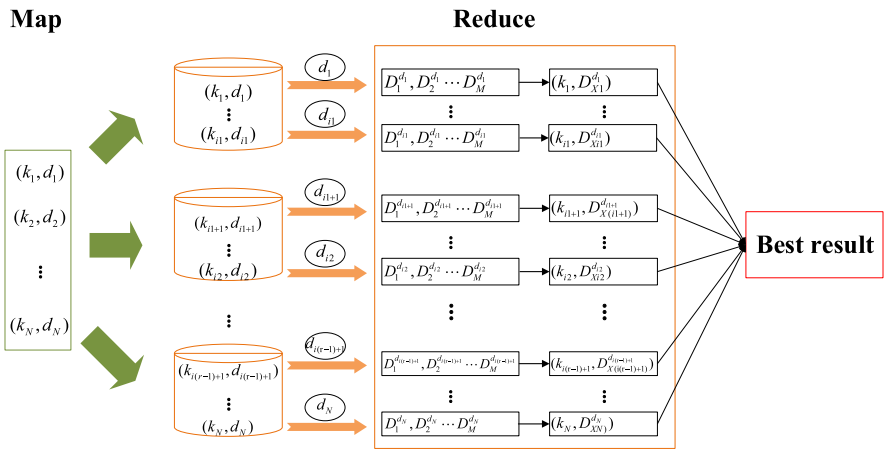


Fig. 1. Diagram of Hadoop Map/Reduce process.

follows: *given a list of items in different sizes to be packed in boxes, how should we pack them such that the number of boxes required is minimum*? The bin-packing problem is NP-complete [13] and therefore efficient heuristic algorithms are needed to solve it. Of these, a simple but efficient one is to sort the items according to their sizes from the largest to the smallest, and then pack them one by one in boxes. It has been shown that for items with regular shapes like balls or cubes, this algorithm is efficient and achieves close to optimal performance.

However, situations do occur where the shapes of the items to be packed and the packing boxes are irregular and different. This would make the above item-sorting algorithm less efficient, and more efficient algorithms are required. For example, the RWA problem is a typical bin-packing problem if we consider different *waveplanes* as boxes and all *lightpath requests* as items [13]. Here, the waveplane is defined as a capacity topology that contains a single wavelength on each links, and all the links in the topology have the same wavelength. A lightpath request is defined as a request to set up a lightpath between a pair of source and destination nodes with one wavelength occupied on all the traversed links. Because the route of a lightpath is not pre-determined, this implies that the "shape" and "size" of the lightpath as an item are irregular. Therefore, efficiently packing these items becomes much more challenging compared to the original bin-packing problem.

In the literature, the waveplane-based algorithm has been shown to be efficient to pack the lightpaths when compared to other algorithms like shortest-path routing and k-shortest path routing with first-fit wavelength assignment [13]. However, even for the most efficient waveplane-based algorithm, we notice that the sequence in which the lightpaths are provisioned can significantly affect the performance in terms of the number of wavelengths used.

One way to mitigate the effect of having different lightpath service provisioning sequences would be to shuffle the lightpath requests randomly many times to generate different sequences. Then, for each of them, the waveplane-based algorithm is implemented to set up lightpaths one by one. Finally, a sequence that performs best is selected to be the final solution of the RWA problem. It is observed that the performance of this shuffling process is closely related to the number of sequences evaluated, i.e., better performance can generally be achieved if more sequences are considered. However, the obvious cost of this is that a longer time is needed to evaluate more sequences, and this can become unacceptably high when the network size is large.

To overcome the above computing difficulty, we consider a parallel optimization mechanism as these shuffled sequences are independent from each other. Specifically, we let different machines run the waveplane-based algorithm for different shuffled sequences in parallel. We implement this mechanism on a Hadoop-based parallel computing system. Fig. 1 illustrates how a heuristic algorithm is parallelly executed multiple times, each for a shuffled sequence. The detailed steps of this process are as follows:

Step 1: The Map function of the Hadoop system generates a total of N key–value pairs (K_i, d_i), for each of which K_i and d_i are both set to be i, i.e., $K_i = d_i = i$.

Step 2: The generated key–value pairs are forwarded to r Reduce functions of the Hadoop system to conduct parallel computing with each Reduce function assigned with N/r key–value pairs.

Step 3: For each key–value pair (K_i, d_i), the list of lightpath requests is randomly shuffled M times with d_i as a random seed to generate M shuffled sequences.

Step 4: For each shuffled sequence, the waveplane-based heuristic algorithm is run to generate M results, from which the best result, denoted by $D_{X_j}^{d_i}$, is chosen.

Step 5: Further compare the best result of each Reduce function to obtain a final optimum, i.e., the "best result".

Since there are a total of N key–value pairs, and for each of the pairs, M shuffled sequences are evaluated, the whole process evaluates a total of $N \cdot M$ shuffled sequences to obtain the final best result. The time taken by the optimization process equals the time taken by each Reduce function if it is executed on a single machine, which is $1/r$ time compared to the case that implements the whole optimization process on a single machine.

In addition to the RWA problem, there are many other problems [2–4] in optical networks that belong to the category of the bin-packing problem and whose heuristic algorithms can also employ the proposed decomposition mechanism for parallel optimization. In this study, we also representatively consider the problem of energy minimization in an IP over WDM network [2], in which the packed item is an IP traffic flow between a node pair and the packing box is a waveplane.

3.2. Meta-heuristic

A meta-heuristic is a high-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computing capacity. The well-known meta-heuristics include simulated annealing (SA), tabu search (TS), iterated local search (ILS), variable local search (VNS), greedy randomized adaptive search procedures (GRASP), etc. The optimization performance of a meta-heuristic is often influenced by its choice of initial value, which is further related to a random seed used in the optimization process. Some seeds may guide the meta-heuristic algorithms to fall into merely local optima, while other seeds may guide the meta-heuristic algorithms to the global optimum. When more seeds are considered, we have a better chance to reach the global optimum. Although different initial values can be chosen to reduce their impact on the optimization performance, this is at the cost of a longer optimization time. On the other hand, since the

optimization processes for the different initial values are independent of each other, we can implement parallelization for this. Specifically, for each initial value, an independent optimization process is implemented to find a sub-optimal solution, which can significantly reduce the time taken for the whole optimization process. Note that this parallelization is essentially identical to the previous optimization process for the shuffled demand requests.

We can also use Fig. 1 to illustrate how a meta-heuristic algorithm can be parallelly executed multiple times, each for a different initial value. The parallelization steps are the same as those for the heuristic scenario except that in Step 4, in which instead of each shuffled demand sequence, the meta-heuristic algorithm is run to generate a sub-optimal result based on each initial value. Moreover, since for each key–value pair, M initial values are evaluated, the whole optimization process considers $M \cdot N$ initial values to get a final optimum result. Because there are r machines executing the meta-heuristic algorithm with different initial values, the Hadoop system takes $1/r$ time to complete the whole optimization process compared to the case where the optimization process is implemented on a single machine.

3.3. MILP

MILP is often employed to model NP-hard/complete problems. Although these problems cannot be solved for large sizes, their optimal solutions can still be found when the problem size is well controlled and sufficient computing power is available. For this, we can also employ parallel computing to increase the computing power for solving an MILP model. As a possible parallelization strategy, we may first split the feasible region of an optimization problem into multiple sub-regions as shown in Fig. 2, and then assign them to different machines to find local optimal solutions in parallel for each of the sub-regions. Once all the local optimal solutions are found, we compare them to obtain a globally optimal solution. Clearly, the computing efficiency of this decomposition mechanism is proportional to the number of sub-regions which are being parallelly considered, or the number of machines employed in the whole optimization process. The more the sub-regions are split, the faster a globally optimal solution may be found. There are different ways to decompose a feasible region. Here, taking the branch-and-bound method as an example, we describe a high-level decomposition mechanism as follows.

Step 1: Given a problem set S, an MILP problem, and the number of computing units N, initialize $S = \varnothing$ and add the MILP problem to S.

Step 2: Get the first problem P in S and delete it from S.

Step 3: Do linear programming (LP) relaxation to remove all the variable integrality restrictions for P and solve it.

Step 4: Check the obtained solution. If all the integer variables are exactly integer, exit; otherwise, move to the next step.

Step 5: Select a variable x_i that is restricted to be integer, but whose value obtained in the LP relaxation is fractional; remove x_i by adding the restrictions $x \leq \lfloor x_i \rfloor$ and $x \geq \lceil x_i \rceil$ to P to generate two sub-problems, respectively, where x is a real variable; then add these two sub-problems to S.

Step 6: Check the size of S. If the size equals to N, exit and output S; otherwise, go to Step 2.

Here, we use the branching variable selection to select a variable in Step 5 [27]. To indicate the quality of a candidate variable, score of each variable is used to measure its effectiveness, and the candidate with the highest score is picked to branch on.

3.4. Machine learning

In recent years, machine learning (ML) has emerged as a promising technique for optimization and big data processing. It is broadly

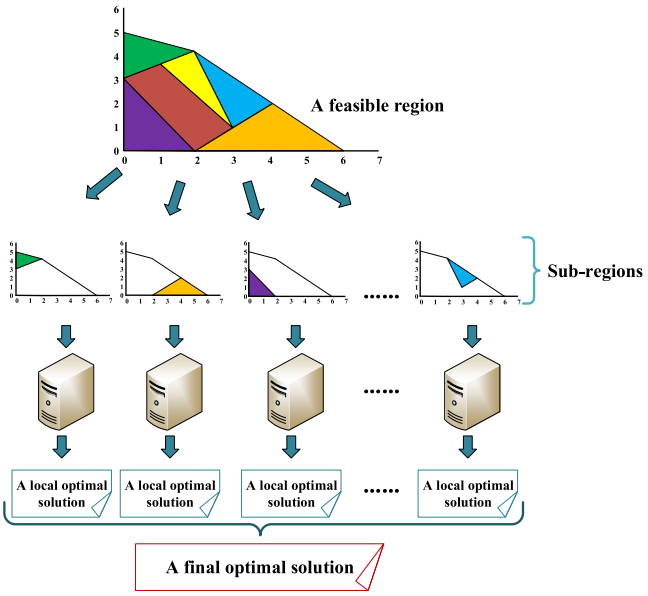


Fig. 2. Diagram of parallelization for solving an MILP model.

divided into three categories: supervised learning, unsupervised learning, and reinforcement learning. To realize massive learning, cloud computing is often employed for very large computing capacity. To demonstrate how parallel computing can enhance learning efficiency, we consider a machine-learning application based on a naive Bayesian classifier, which belongs to the category of supervised learning. Specifically, we employ a naive Bayesian classifier to solve the routing problem in the circuit-switched network (note that a wavelength-routed optical network belongs to this category) [28]. For efficient routing, we first learn the service provisioning process in the context of different network status to generate a naive Bayesian classifier, and then use this classifier to select a route for service provisioning which is expected to achieve the lowest blocking probability. It was shown that the proposed approach can outperform the conventional best routing algorithm, i.e., the least-loaded routing algorithm. Note that there are two types of least-loaded routing algorithms in the literature. In the first type of least-loaded routing algorithm, the bottleneck link of each route is first determined, and then the route with the largest bandwidth on the bottleneck link is selected to establish a service connection [29]. In the second type of least-loaded routing algorithm, the links without sufficient bandwidth are first removed, and then the congestion level of each link is employed as a weight to find a shortest route between a pair of nodes. The route thus found is then used to establish a service connection [30]. Because the second type of algorithm can perform better, it is employed as the least-loaded routing algorithm for the ML study in [28]. Fig. 3 illustrates how the naive Bayesian classifier-assisted least-loaded routing algorithm is implemented in the network control plane, with a detailed learning process as follows:

Step 1: The ML-processor implements an initial learning process to obtain an initial set of classifier parameters.

Step 2: When a service request arrives, the network controller checks the ML-processor to see whether the service connection can be provisioned; if so, the ML-processor feeds back the route information to the network controller and the latter establishes the service connection.

Step 3: After the service connection is established, the ML-processor performs an online learning based on the information of the service request and the network status when the service request arrives, and updates its classifier parameters. The ML-processor also forwards the information of the network status and the service connection to a network state database to update the historical network state information.

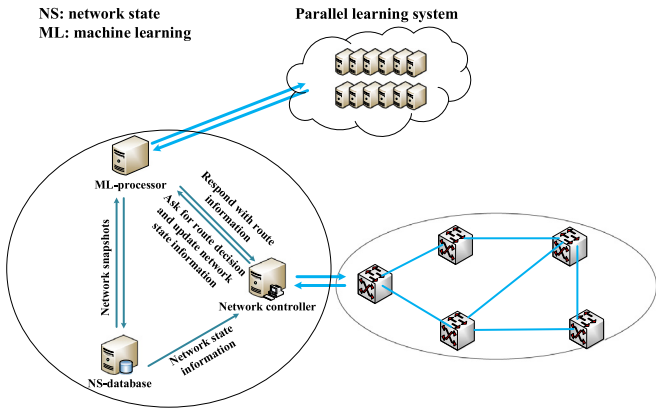


Fig. 3. Learning process of naive Bayesian classifier-assisted least loaded routing algorithm.

Step 5: When a certain number of service connections arrive and the ML-processor is sufficiently trained, the ML-processor outputs a final service connection blocking probability and the updated classifier parameters.

To enable parallel learning, Fig. 4 illustrates a decomposition process, whose details are as follows:

Step 1: The master unit (a machine) splits a learning task into multiple sub-tasks and distributes them to N slave units (machines). Here, if a learning task is large (i.e., a large number of service arrival events need to be learned), multiple rounds of iterative learning are required.

Step 2: Upon receiving a learning task, each slave unit starts to learn independently and in parallel.

Step 3: Once the learning task of a slave unit is completed, it feeds back its output (including the number of blocked requests and the updated classifier parameters) to the master unit.

Step 4: Once the master unit receives the feedback, it updates the total number of blocked requests in the entire learning process and updates the classifier parameters using the ones from the slave unit that has the lowest blocking probability. These updated classifier parameters are then distributed to all the slave units.

Step 5: Once a certain total number of requests have been learned, the master unit terminates the learning process and calculates the overall network blocking probability.

4. Miniaturized parallel computing system

To support this parallel optimization, we design a miniaturized parallel computing system based on the Hadoop software platform. In this section, we first describe the operating principle of the system, which consists of sub-problem decomposition and distribution, and synthesis of the output from different computing units. We next describe a prototype of the system.

4.1. Operation principle

In a Hadoop system, we divide all computing units into one master unit and multiple slave units. Here, the computing unit can be a regular computer or a powerful one. When a powerful one is employed as a basic computing unit, then the whole parallel computing system would become even more powerful. The master unit is the core of the whole system and is responsible for (1) decomposing the optimization problem, (2) distributing the sub-problems, and (3) collecting and synthesizing the solutions to the sub-problems. It also coordinates the computing tasks on each slave unit to balance their workload and maximize the computing efficiency of the whole system. Each slave unit functions to (1) receive a sub-problem from the master unit, (2) solve the sub-problem, and (3) feedback the solution of the sub-problem to the master unit. It also periodically feeds back its own real-time workload to the master unit. In addition, to fully utilize the computing resources of the system, a master unit can also work as a slave unit to solve sub-problems as required.

Fig. 5 illustrates the parallel computing process. First, the master unit splits an optimization problem into multiple sub-problems, and then distributes them to multiple slave units based on their workloads. Once a slave unit receives a sub-problem, it solves the sub-problem and feeds back a final result to the master unit. When the master unit receives the feedback results, it compares all of them to find a globally optimal solution. Specifically, we employ the MapReduce module in the Hadoop system to implement the above process. Upon receiving an optimization problem, the Map function in the master unit decomposes the problem into multiple sub-problems and then distributes them with their related initial data to multiple Reduce functions (each of which is hosted in a slave unit). Each Reduce function solves its own sub-problem and feeds back its result to the master unit. The latter collects all the results and synthesizes them to obtain a globally optimum solution.

Note that for some situations (e.g., in machine learning), the above parallel computing process can be iterative. That is, after the master unit receives a result from all the slave units, it would synthesize the results and generate a new set of sub-problems and forward them to the slave units to repeat the same process. The iterative process will be terminated whenever a certain optimization performance is reached, or a certain number of iterations are performed. For example, in the previous machine learning example with the naive Bayesian classifier, ten rounds of iterative parallel learning are performed to learn 100 million arrival events, in which each round of learning considers 10 million arrival events. When one round of parallel learning is completed, the master unit will synthesize the related classifier parameters according to the previous round of learning and will distribute the new

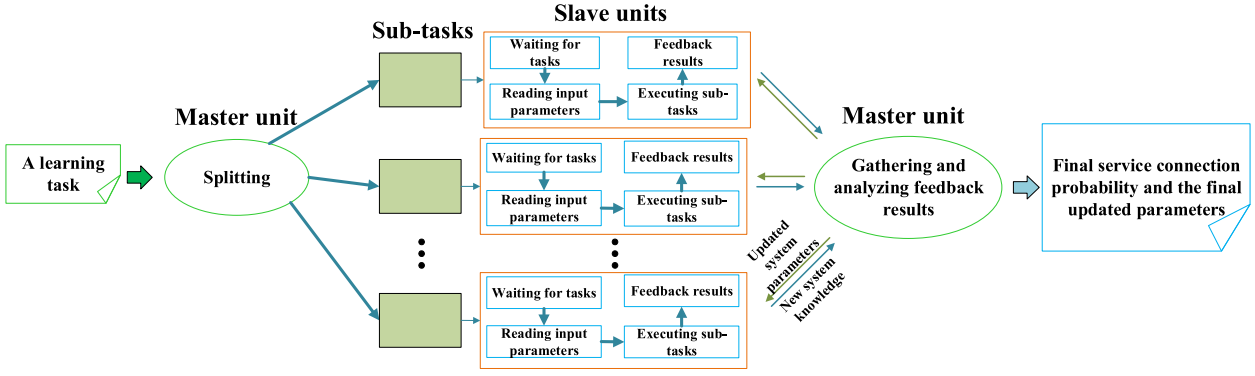


Fig. 4. Diagram of multi-round parallel learning.

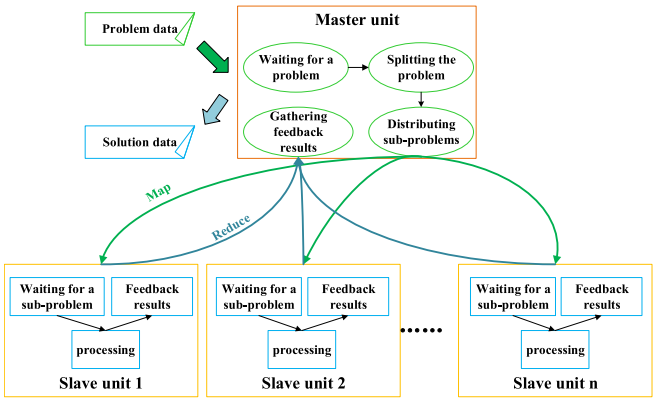


Fig. 5. Diagram of Hadoop-based parallel computing.

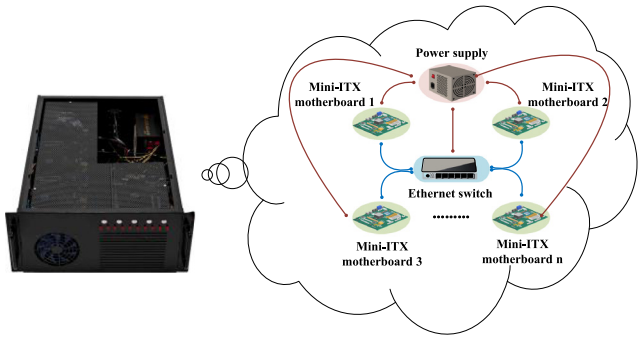


Fig. 6. Prototype and logical layout of the third-generation system.

parameters to each slave unit to start a new round of learning. The entire learning process will stop when a total of 100 million arrival events are learned, based on which the master unit obtains a final service blocking probability and a final set of classifier parameters.

4.2. Prototype

We have prototyped the above parallel computing system, in which various types of machines can be supported, e.g., an ordinary desktop, a CubieBoard with ARM processor [31], etc. In the first-generation parallel computing system, we employed multiple ordinary desktops to build a Hadoop system, which however occupies a large space and suffers from a complicated network setup. To shrink the system, in the second generation, we used a CubieBoard to replace the ordinary desktop, which allows us to host the whole system with 10 CubieBoards in a 4U server box (whose size is 177.8 (height) × 428.6 (width) × 500 (length) mm). However, the processor of CubieBoard in this generation is low-end, not able to provide high-computing capacity. Therefore, to further increase the computing capacity, we developed the third-generation system, in which each CubieBoard is replaced by an Intel H81 Mini-ITX motherboard which supports high-end processors. Fig. 6 shows the hardware prototype and logical layout of this generation of system, which is hosted in a 5U server box with the size of 222.5 (height) × 430 (width) × 650 (length) mm.

Table 2 shows the hardware configuration of the third-generation system. The size of each Intel H81 Mini-ITX motherboard is 170 × 170 mm, which is cheaper and more compact and energy-efficient than an ordinary computer. On the motherboard, the adopted CPU is an Intel® Core™ i7-4790K processor while more advanced CPUs can also be supported. In addition, a 256-GB solid state disk and a 16-GB memory module are embedded on each motherboard. A 16-port Gigabit Ethernet switch is used to connect all the Mini-ITX motherboards to form a computing cluster. The power supply of the system complies with the ATX 12 V and EPS 12 V standards, and can maximally deliver 1050-W power. Finally, two fans are configured to cool the system.

Table 2
Hardware components of the system.

Part	Specification
CPU	Intel® Core™ i7-4790K processor
Motherboard	Intel H81 Mini-ITX motherboard
Hard disk drive	256-GB solid state disk
Memory module	16 GB
Switch	A 16-port Gigabit Ethernet switch
Power supply	1050 W
Cooling system	Two fans

5. Experiment and results

We employ the parallel optimization system developed to validate the performance (in terms of optimality and computing time) of the different parallel optimization methods, including heuristic, meta-heuristic, MILP, and machine learning. To provide a benchmark, we also evaluate the performance based on a single machine, which is one of the computing units in the parallel optimization system.

Four test networks are considered for the performance evaluation, including a 50-node, 100-link network (randomly generated), the 11-node, 26-link COST239 network, the 14-node, 21-link NSFNET network, and 24-node, 43-link US backbone network (USNET). Fig. 7 shows the topologies of COST239, NSFNET, and USNET. The random network is generated in the following manner. First, we generate a ring network containing a specific number (e.g., 50) of nodes. Second, if the current number of links is smaller than a predefined number (e.g., 100) of links, we randomly select a pair of nodes that are not connected yet and add a link between them. We continue this link-adding process until the required number of links is reached.

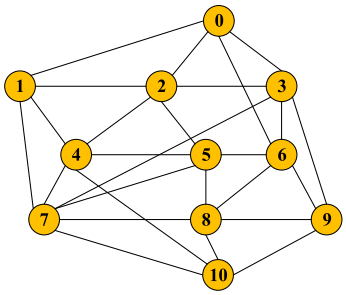
5.1. Heuristic

For the heuristic, we representatively consider two bin-packing optimization problems, which are [1] the routing and wavelength assignment (RWA), and [2] the energy minimization in IP over WDM network.

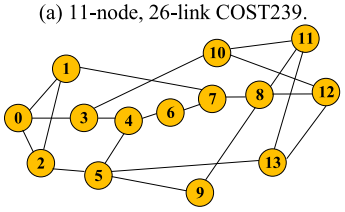
5.1.1. Routing and wavelength assignment (RWA)

We solve the RWA problem for the random (50-node, 100-link) network, in which there are a total of 220 requests [1]. We shuffle these requests 10,000 times, and then run the RWA algorithm for each shuffled demand sequence. Fig. 8 shows the maximum number of wavelengths required by each shuffled demand sequence. It is noted that the number of wavelengths varies in the range of [37, 39] for the different shuffled demand sequences. This therefore confirms that the number of required wavelengths is related to the provisioning sequence of lightpath services and it is important to evaluate multiple shuffled sequences for optimal performance.

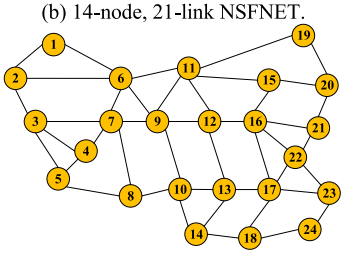
We also evaluate the computing efficiency of the parallel computing system in running the RWA algorithm for multiple shuffled demand sequences in comparison with a single machine [13]. Fig. 9 shows the computing times taken by the parallel computing system and a single machine when a total of 10,000 and 100,000 shuffled demand sequences are considered, respectively. The legend “Single” corresponds to a single machine, and the legend “Parallel” corresponds to the parallel computing system. We see that the parallel computing system is efficient to significantly shorten the optimization time for evaluating different shuffled demand sequences in comparison with a single machine. Specifically, up to 74% and 77% time is saved by the parallel computing system for evaluating 10,000 and 100,000 shuffled demand sequences, respectively. Moreover, the time saving is more significant when a larger number of shuffled demand sequences are considered.



(a) 11-node, 26-link COST239.



(b) 14-node, 21-link NSFNET.



(c) 24-node, 43-link USNET.

Fig. 7. Test networks.

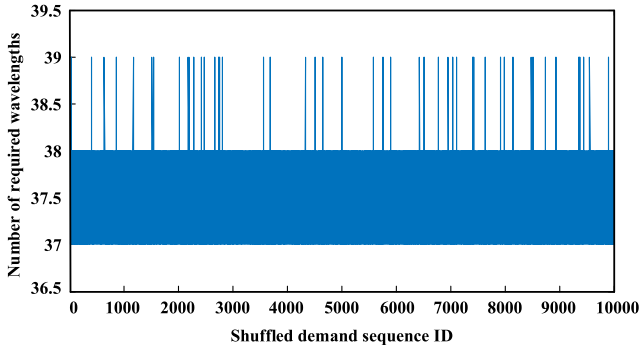


Fig. 8. Numbers of wavelengths required by different shuffled demand sequences.

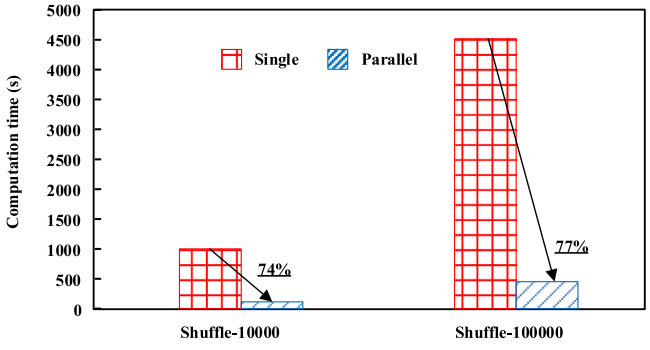


Fig. 9. Time taken by the parallel computing system and a single machine for evaluating different number of shuffled demand sequences.

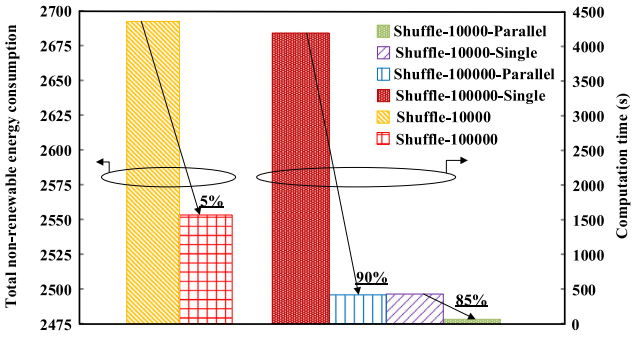


Fig. 10. Total non-renewable energy consumed and computing time (USNET).

5.1.2. Energy minimization in IP over WDM network

We also evaluate the optimization efficiency of the parallel computing system based on the problem of energy minimization in an IP over WDM network [14]. USNET is employed for this evaluation, in which each node pair is assumed to have traffic demand varying in the range of [10,30] Gb/s. Details on the weather conditions, the geographic locations, and the renewable energy outputs of each network node in the USNET network can be found in [2]. The traffic grooming heuristic algorithm of [2] is used for this evaluation.

Fig. 10 compares the total non-renewable energy consumed and the total computing time taken by the parallel computing system in comparison with a single machine when 10,000 to 100,000 shuffled demand sequences are considered. Again, the legends "Single" and "Parallel" corresponds to the results of a single machine and the parallel computing system, respectively. We note that considering a larger number of shuffled demand sequences helps improve energy efficiency. Specifically, compared to 10,000 shuffled demand sequences, 100,000 shuffled demand sequences can reduce the total non-renewable energy consumed by 5%. Moreover, the parallel computing system is computationally efficient to significantly reduce the optimization time compared with a single machine, i.e., up to 85% and 90% computing time can be reduced, respectively, when evaluating 10,000 and 100,000 shuffled demand sequences.

5.2. Meta-heuristic

For the meta-heuristic, we consider the problem of network resource allocation in a virtualized EON. In [32], we propose a new virtual EON (VEON) service provisioning scheme which includes the stages of spectrum assignment, spectrum trading, and spectrum purchasing. One meta-heuristic algorithm and two heuristic algorithms are proposed respectively for the three stages to minimize the total cost paid by VEON clients. Specifically, in the stage of spectrum assignment, a simulated annealing (SA)-based virtual optical network embedding (VONE) meta-heuristic algorithm is developed to assign spectra for the VEONs. The performance of this meta-heuristic algorithm is dependent on the random seed chosen, which further impacts the overall performance of the VEON service provisioning scheme. To achieve a good VONE performance, we run the meta-heuristic algorithm based on multiple random seeds and employ the parallel computing system developed to perform this multi-seed evaluation. Specifically, we let each machine in the parallel computing system run the SA-based VONE algorithm based on a specific random seed. COST239 is employed for performance evaluation, in which three modulation formats, i.e., BPSK, QPSK, and 8-QAM, are considered for setting up virtual links based on their physical distances. We also assume that there are 50 VEONs, each of which has $N \in \{6,7\}$ virtual nodes and $L \in \left[N-1, N\cdot(N-1)/2\right]$ virtual links. The lifetime of the VEONs is divided into 50 time slots (TSs). The average traffic demand on each virtual link is 300 Gb/s.

To verify the efficiency of the SA-based VONE meta-heuristic algorithm, Fig. 11 shows the performance improvement with increasing

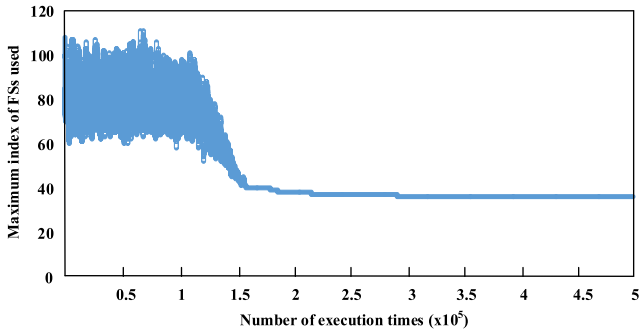


Fig. 11. A simulated annealing (SA) process for the VONE problem (COST239).

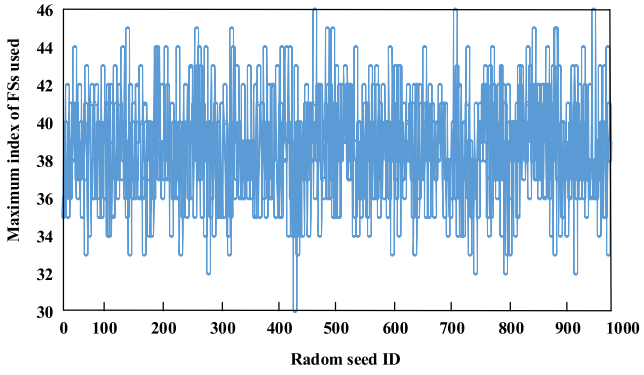


Fig. 12. Maximum index of FSs used for different random seeds (COST239).

number of SA execution times (random seed = 10). We note that with increasing number of execution times, the maximum index of FSs used fluctuates at the beginning but converges at the end with the smallest number of FSs used, thereby verifying the effectiveness of the meta-heuristic algorithm.

To show how different random seeds can impact the optimization performance of the SA-based VONE meta-heuristic algorithm, we also run the meta-heuristic algorithm based on 1000 different random seeds and show the results in Fig. 12. We see that the results are closely related to the random seeds selected, and vary between 30 and 46 for different random seeds. This confirms the influence of the random seed selection on the optimization performance.

Fig. 13 compares the smallest maximum index of FSs used and the total computing time taken by the parallel computing system in comparison with a single machine when 1000 to 10,000 random seeds are considered. The legend "RD-X" corresponds to the case when evaluations are done with X random seeds. We see that considering a larger number of random seeds can reduce the maximum index of FSs used. Compared to the case of 1000 random seeds, 10,000 random seeds can decrease the maximum index of FSs used by up to 11%. We also note that compared to a single machine, the implementation using a parallel computing system is more efficient and reduces the optimization time by up to 39% and 41% for 1000 and 10,000 random seeds, respectively.

In addition, the computation time of each slave unit with 10 random seeds is shown in Fig. 14. We can see that the longest computation time is 10,480 s and the shortest computation time is 7492 s. The average computation time is 9141 s, with a standard deviation of 980 s. The load of each slave unit does not seem to be fully balanced in the current system. Properly balancing the load on each slave unit would therefore be important for further study. Note that, in addition to the meta-heuristic scenario, the issue of load imbalance of the slave units also exists in the other optimization approaches and should be studied further for these approaches as well.

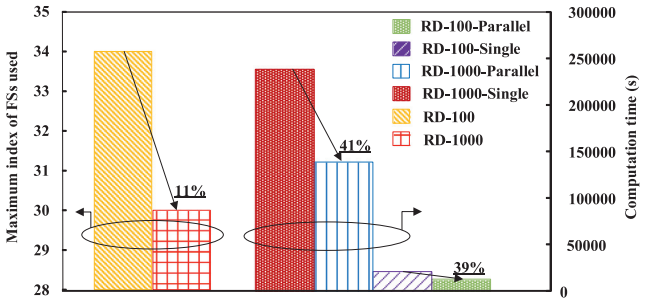


Fig. 13. Maximum index of FSs used and computation time (COST239).

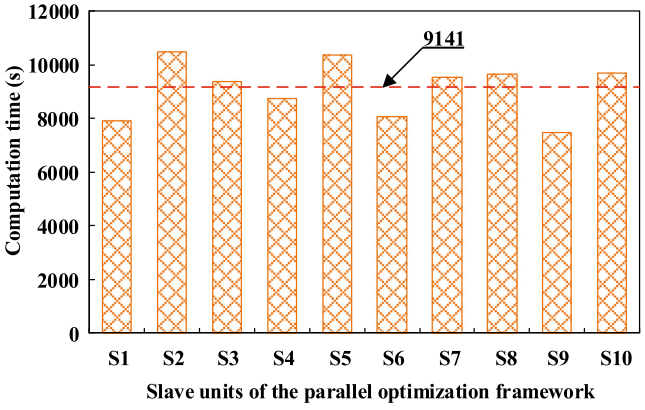


Fig. 14. Computation times of different slave units.

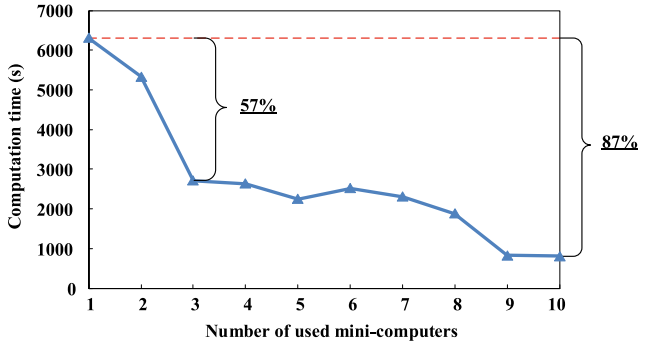


Fig. 15. Computing time of solving MILP problems (NSFNET).

5.3. MILP

For the MILP-based optimization, we employ the optimization problem of [33] to evaluate the computing efficiency of the parallel optimization mechanism and system developed here. This focuses on minimizing the energy consumption of an IP over WDM network. In [33], we formulate an MILP model to minimize the total energy consumption contributed by various network components including IP routers, optical amplifiers, and WDM transponders. We verify the parallel optimization approach by employing the commercial software package Gurobi (parallel version 9.12) [34] to solve this MILP problem. The MIPGAP for the MILP solution is set to be 0.1%. NSFNET is employed for this performance evaluation, in which the average traffic demand between each node pair is 120 Gb/s.

Fig. 15 shows the computing time taken by the parallel computing system when different numbers of minicomputers are turned on. The

number of threads in each minicomputer is set to be 5. We note that turning on more minicomputers saves more computing time as compared to when a single machine is used. Overall, there is a linear relationship between the computing time taken and the number of minicomputers used. Specifically, the system takes 6309 s to solve the MILP problem when there is only a single machine. However, when the number of minicomputers increases to 3, the computing time reduces to 2716 s, which is a saving of about 57%. When all the minicomputers are turned on, the computing time further reduces to 811 s, which corresponds to an 87% saving compared to the case of a single machine.

5.4. Machine learning

For machine learning, we have employed the parallel computing system developed to perform parallel learning in the context of a naive Bayesian classifier-assisted least-loaded routing algorithm [28]. In that study, we parallelly simulated and learned up to 100 million service arrival events. The parallel learning approach can significantly improve the service blocking performance to outperform the conventional best routing algorithm, i.e., the least-loaded routing algorithm. It also significantly reduces the computing time (by up to 90%) in comparison to when a single machine is used. Interesting readers can further refer to [28]. This once again confirms the efficiency of the proposed parallel learning mechanism and system.

6. Conclusion

We developed parallel optimization mechanisms for different optimization approaches, including heuristic, meta-heuristic, MILP, and machine learning. To verify the effectiveness of these mechanisms, we developed a miniaturized parallel computing system by integrating regular minicomputers. Based on this system, the effectiveness of the proposed parallel optimization mechanisms was verified in the context of various optimization problems in the optical network. The results show that the proposed parallel optimization mechanisms are effective to reduce computation times and improve the solution quality for many different optimization approaches including heuristic, meta-heuristic, MILP, and machine learning. Though this study has considered only a few representative optimization problems, the proposed mechanisms and the developed parallel computing system are generic enough to also solve other similar optimization problems in a wide variety of similar categories.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] G. Shen, C. Wu, J. Dong, An almost-optimal approach for minimizing the number of required wavelengths for large-scale optical networks, in: Proc. Opt. Fiber Commun. Conf. Expo. Nat. Fiber Opt. Eng. Conf., 2013, pp. 1–3.

[2] G. Shen, Y. Lui, S.K. Bose, Follow the sun, follow the wind lightpath virtual topology reconfiguration in IP over WDM network, IEEE/OSA J. Lightwave Technol. 32 (11) (2014) 2094–2105.

[3] C. Wang, G. Shen, S.K. Bose, Distance adaptive dynamic routing and spectrum allocation in elastic optical networks with shared backup path protection, IEEE/OSA J. Lightwave Technol. 33 (14) (2015) 2955–2964.

[4] Y. Li, H. Dai, G. Shen, S.K. Bose, Adaptive FEC selection for lightpaths in elastic optical networks, in: Proc. Opti. Fiber Commun. Conf., 2014, pp. 1–3.

[5] S. Zhang, S. Zhang, X. Chen, X. Huo, Cloud computing research and development trend, in: Proc. Second Int. Conf. on Future Networks, 2010, pp. 93–97.

[6] A. Middleton, A. Chala, Hpcc Systems: Introduction to Hpcc (High-Performance Computing Cluster), White paper, LexisNexis Risk Solutions, 2011.

[7] Hadoop, 2021, [Online]. Available: https://hadoop.apache.org. [Accessed: Aug. 1, 2021].

[8] Spark, 2021, [Online]. Available: https://spark.apache.org. [Accessed: Aug. 1, 2021].

[9] T.S. Hukkeri, G. Shobha, S.M. Phal, J. Shetty, H.R. Yatish, N. Mohammed, Massively scalable image processing on the HPCC systems big data platform, in: Proc. 3rd Int. Conf. Sof. Eng. Infor. Mana., 2021, pp. 26–31.

[10] L. Xu, A. Apon, F. Villanustre, R. Dev, A. Chala, Massively scalable parallel kmeans on the HPCC system platform, in: Proc. 4th Int. Conf. Comp. Syst. Infor. Technol. Sust. Sol., 2019, pp. 1–8.

[11] V.M. Herrera, T.M. Khoshgoftaar, F. Villanustre, B. Furht, Random forest implementation and optimization for big data analytics on LexisNexis's high performance computing cluster platform, J. Big Data 6 (1) (2019) 1–36.

[12] Q. Qiu, Q. Wu, M. Bishop, R.E. Pino, R.W. Linderman, A parallel neuromorphic text recognition system and its implementation on a heterogeneous high-performance computing cluster, IEEE Trans. Comput. 62 (5) (2013) 886–899.

[13] G. Shen, Y. Li, L. Peng, Almost-optimal design for optical networks with hadoop cloud computing: ten ordinary desktops solve 500-node, 1000-link, and 4000-request RWA problem within three hours, in: Proc. 15th Int. Conf. Trans. Opt. Netw., 2013, pp. 1–4.

[14] Y. Li, G. Shen, B. Chen, M. Gao, X. Fu, Applying Hadoop cloud computing technique to optimal design of optical networks, in: Proc. Asia Commun. Photon. Conf., paper ASu3H.3, 2015.

[15] Y.C. Shen, C.H. Hsu, S.H. Hsieh, Integrated genetic algorithms and cloud technology to solve travelling salesman problem on Hadoop, in: Proc. IEEE Int. Conf. on Cloud Comp. Tech. and Sci., 2012, pp. 566–569.

[16] G. Yildirim, I.R. Hallac, G. Aydin, Y. Tatar, Running genetic algorithms on Hadoop for solving high dimensional optimization problems, in: Proc. IEEE 9th Int. Conf. on Appli. of Infor. and Commu. Tech., 2015, pp. 12–16.

[17] Y. Yu, X. Bu, K. Yang, H.K. Nguyen, Z. Han, Network function virtualization resource allocation based on joint benders decomposition and ADMM, IEEE Trans. Veh. Technol. 69 (2) (2020) 1706–1718.

[18] S.A. Ludwig, MapReduce-based optimization of overlay networks using particle swarm optimization, in: Proc. 2014 Annu. Conf. Gen. Evol. Comp., 2014, pp. 1031–1038.

[19] M. Sherar, F. Zulkernine, Particle swarm optimization for large-scale clustering on apache spark, in: Proc. IEEE Sym. Ser. on Compu. Intell., 2017, pp. 1–8.

[20] H. Chen, P. Chang, Z. Hu, H. Fu, L. Yan, A spark-based ant lion algorithm for parameters optimization of random forest in credit classification, in: Proc. IEEE Inf. Tech. Netw. Elec. Autom. Con. Conf., 2019, pp. 992–996.

[21] P. Liu, S. Ye, C. Wang, Z. Zhu, Spark-based parallel genetic algorithm for simulating a solution of optimal deployment of an underwater sensor network, Sensors 19 (12) (2019) 2717.

[22] J. Yuan, An anomaly data mining method for mass sensor networks using improved PSO algorithm based on spark parallel framework, J. Grid Comput. 18 (2) (2020) 251–261.

[23] D.S. Terzi, R. Terzi, S. Sagiroglu, Big data analytics for network anomaly detection from netflow data, in: Proc. Int. Conf. Comp. Sci. Eng., 2017, pp. 592–597.

[24] G.P. Gupta, M. Kulariya, A framework for fast and efficient cyber security network intrusion detection using apache spark, Procedia Comput. Sci. 93 (2016) 824–831.

[25] F. Morales, M. Ruiz, L. Gifre, L.M. Contreras, V. López, L. Velasco, Virtual network topology adaptability based on data analytics for traffic prediction, IEEE/OSA J. Opt. Commun. Networking 9 (1) (2017) A35–A45.

[26] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, K. Li, A parallel random forest algorithm for big data in a spark cloud computing environment, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2016) 919–933.

[27] T. Achterberga, T. Kocha, A. Martinb, Branching rules revisited, Oper. Res. Lett. 33 (1) (2005) 42–54.

[28] L. Li, Y. Zhang, S.K. Bose, M. Zukerman, G. Shen, Naïve Bayes classifier-assisted least loaded routing for circuit-switched networks, IEEE Access 7 (2019) 11854–11867.

[29] A. Mokhtar, M. Azizoglu, Adaptive wavelength routing in all-optical networks, IEEE/ACM Trans. Netw. 6 (2) (1998) 197–206.

[30] G. Shen, S.K. Bose, T.H. Cheng, C. Lu, T.Y. Chai, Efficient heuristic algorithms for light-path routing and wavelength assignment in WDM networks under dynamically varying loads, Comput. Commun. 24 (3–4) (2001) 364–373.

[31] CubieBoard, 2021, [Online]. Available: https://www.http://cubieboard.org/. [Accessed: Aug. 3, 2021].

[32] S. Ding, Z. Jiang, S.K. Bose, G. Shen, Cost-minimized virtual elastic optical network provisioning with guaranteed QoS, China Commun. 18 (9) (2021) 148–166.

[33] G. Shen, R.S. Tucker, Energy-minimized design for IP over WDM networks, IEEE/OSA J. Opt. Commun. Networking 1 (1) (2009) 176–186.

[34] Gurobi, 2021, [Online]. Available: https://www.gurobi.com/. [Accessed: Aug. 1, 2021].