Robust Path Selection Schemes under Uncertain Network Conditions

Ravindra Sandaruwan Ranaweera

Department of Computer and Network Engineering

The University of Electro-Communications

A thesis submitted for the degree of

Doctor of Philosophy

March 2019

APPROVED BY SUPERVISORY COMMITTEE:

Chairperson: Prof. Kitsuwan Nattapong

Member: Prof. Naoto Kishi

Member: Prof. Yasushi Yamao

Member: Prof. Eiji Oki

Member: Prof. Minoru Terada

Date of the final-defense: 2019/02/04

論文の和文概要

ネットワーク状態には、トラフィック要求、ネットワークデバイスまたはリンクの障害、およびノード負荷が含まれている。ユーザーが満足するサービスを提供しながら良好なサービス品質を達成するためには、ロバスト的かつ効率的な経路を選択する必要がある。

本論文は、不確実なネットワーク条件でロバストな経路選択方式を提供する手法を提案する。具体的には、トラフィック需要とリンク障害の不確定性を持つインターネットプロトコル（IP）ネットワークのためのリンク重み計算方式と、ノード負荷を考慮した経路選択方式について提案する。また、それぞれ提案方式の効果をシミュレーション・実験を用いて確認し、最後に論文のまとめと今後展望について説明する。

Abstract

The constant evolution of network technologies pose a challenge in network performance management due to complicated and uncertain network conditions. Network conditions include traffic demand, failures in network devices or links, and node load. In order to achieve a desired quality of service while providing a satisfying service to customers, networks must be able to select paths robustly and efficiently. This thesis provides robust path selection schemes under uncertain network conditions. This thesis focuses on the uncertain network conditions of traffic demands, link failures, and node load, since they strongly affect network services.

In the first part of the thesis, a path selection scheme for Internet Protocol (IP) networks with traffic demand and link failure uncertainty is presented. Open Shortest Path First (OSPF) is widely used as a routing protocol in IP networks. OSPF selects the shortest path between source and destination pairs. To determine shortest paths, OSPF requires to assign link weight for each link in the network. If link weights are not optimal, network traffic may concentrate to one link and it may increase the packet drop rate, reducing the quality of service. Implementing the most appropriate set of link weights is a challenging traffic engineering problem in OSPF networks.

Start-time Optimization (SO) is a commonly used link weights optimization scheme. However, it does not consider network condition changes caused by network failures, such as link failure, nor traffic demand changes. Preventive Start-time Optimization (PSO) on the other hand considers network changes caused by link failures preventively and calculates an optimal set of link weights. SO and PSO

were introduced for scenarios where traffic demand is exactly known. However, estimating or measuring traffic demand exactly is a difficult task for network operators. These link weight optimization methods may not be fully applicable in a context where the traffic demand often fluctuates. In order to handle traffic demand fluctuations dynamically, the path selection scheme presented in this thesis uses a so-called hose model. In the hose model, specifying the total egress and ingress traffic at each node is enough; a traffic demand between each source and destination pair is not required to be specified.

Network operators may have different objectives when configuring routing paths. One network operator may want to configure paths to reduce the network congestion. Another network operator may want to configure paths to reduce the network resource usage. In order to address these objectives, the path selection scheme presented in this thesis is applied to reduce the network congestion and the network resource usage. The network congestion reduction and the network resource reduction problems are formulated as mixed integer linear programming (MILP) problems that consider both traffic demand fluctuation and link failure. To mitigate the complexity of solving the MILP formulations, heuristic approaches are also presented. Simulation results show that the presented scheme, while being robust to traffic demand fluctuations and link failures, is able to reduce the network congestion and the network resource usage.

In the second part of the thesis, a path selection scheme for networks considering node load is presented. A group of nodes or servers connected with links is also a network. In order to communicate between servers in the network, a source server sends data packets to a destination server through the network and the destination server responds to the source server accordingly. As a consequence of the destination server being heavily loaded, data packets will be queued to be processed later making the response time longer. On the other hand, relatively free destination server will respond quickly finishing the communication process in a shorter time. Therefore, response time or delay time between two servers can be used to measure server load.

Apache Hadoop (Hadoop) is a parallel-distributed computing framework and usually consists with tens to thousands of servers. These servers are connected via network to run jobs in a distributed manner. The storage layer of Hadoop, called Hadoop Distributed File System (HDFS) calculates paths between servers based on each server's rack assignment information. Rack assignment is configured by the cluster administrator. HDFS keeps three (by default) or more replicas (copies) of each data block for fault tolerance and availability purposes. When a client wants to fetch non-local data, HDFS selects a server that holds a replica of that data based on the proximity to the client. This source client and multiple destination servers that hold the data, result in multiple paths between the source and destinations. Selecting one destination server to fetch data from can also be said as selecting one path between the source and a destination server. HDFS only considers the distance (hop count) between servers and does not consider server load of the destination when selecting a destination server. This hop count based static behavior of selecting a destination server decreases the overall efficiency of the cluster, wasting valuable cluster resources.

This thesis also presents a path selection scheme based on delay distribution between servers for Hadoop clusters. This scheme selects a destination server by comparing the delay distributions between client-server pairs. The delay distributions are calculated by measuring the round-trip time between server pairs periodically. Our experiments observe that the presented scheme, while being robust to server load and network delay, is able to select the best path resulting shorter data fetch time compared to conventional Hadoop. This reduction in data fetch time will lead to the reduction in job runtime, especially in real-world multi-user clusters where non-local data fetch can happen frequently.

Acknowledgements

This thesis is the summary of my doctoral study at the University of Electro-Communications, Tokyo, Japan. I am grateful to a large number of people who have helped me to accomplish this work.

First of all, I would like to express my sincere gratitude to my advisors, Professor Eiji Oki and Assistant Professor Nattapong Kitsuwan, for their mentor-ship, guidance and encouragements. Especially, by working under the supervision of Professor Oki, I received an invaluable experience that helped me to shape my academic and professional skills.

I would like to thank my previous senior manager at work, Mr. Shuji Ushida for allowing me to start my doctoral study while working full-time at the company.

I would like to express my sincere appreciation to Professor Naoto Kishi, Professor Yasushi Yamao, and Professor Minoru Terada for being part of my judging committee and providing their precious comments to improve this thesis.

Finally, I want to deeply thank my lovely wife, parents, and friends for all the support and caring.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Communication networks (networks) play a critical role in modern software systems. Routing is the main process used to send data from point A to point B by determining a path (route) that exists in the network [1]. In this thesis, the term *routing* is used to refer sending data along the selected path. A stable network is crucial for software systems to provide their intended services without any interruption. However, networks are inherently unpredictable requiring robust and efficient path selection schemes to provide satisfying services to customers.

Network instability is caused by a variety of factors; link failures, router failures, traffic demand fluctuations, node overload, software configuration errors and bugs, malicious attacks, and human errors [1, 2, 7]. One or more of these problems affect the performance of the network. This thesis considers link failures, traffic demand fluctuations, and node load since they strongly affect network services and can be managed by routing schemes easily compared to others. Since router failures can be interpret as multiple link failures, malicious network attacks and errors related to software and human are unmanageable by routing schemes, they are considered as being out of scope of this work.

1.1 Uncertainty of network conditions

In general, a network consists with nodes and links. A link connects two nodes allowing them to exchange data between each other. Data that travel through the network is called traffic in general. A critical function of a network is to carry traffic from one node to another based on routing decisions. Usually, routing decisions are dictated by the requirements of the network in addition to specific goals of a network service provider. Such goals may include how to provide an efficient and fair service so that most users receive an acceptable service rather than few specific users receiving outstanding service. Aforesaid view is partly required because there are finite amount of resources in a network, e.g., link capacity, and a service provider has a responsibility to treat all traffic neutrally [9].

The traffic volume that a source node requests to send to a destination node, is called a traffic demand in Internet Protocol (IP) networks. The set which describes all the demands between every source-destination pair in the network is called the traffic matrix. In real-world networks, traffic demand between all the source-destination node pairs can easily fluctuate. This makes it difficult for network operators or network administrators to know the actual traffic matrix, especially when the network size is large [8]. This uncertain nature of the traffic can impact the path selection decisions of the network adversely affecting the quality of service [1, 10, 11, 39]. Considering traffic demand uncertainty for design and planning of IP networks has recently attracted much attention [10, 11, 13, 14, 15, 40]. Bauschert et al. [10] presented multi-layer and mixed-line-rate network designs for network planning under traffic demand uncertainty. In [11], the authors considered the egress traffic demand at each node to calculate robust paths. Mitra et al. [12] presented a scheme to maximize the revenue under uncertain traffic demands. All of these work stress the importance of robust path selection schemes under uncertain traffic demands. Therefore, it is necessary to study ways to minimize the impact of traffic demand fluctuations.

Emerging software systems rely on the reliability of networks. Unfortunately, due to network failures such as link failures, maintenance windows, router reboots, etc. these services are adversely affected [17, 18]. Among these failures, link failures are considered to be one of the main challenges faced by network operators because link failures occur on a daily basis [17]. Moreover, it is observed that most of the link failures are common and transient. Failures in information networks are common and can result in substantial losses [19]. Recovery from link failures in IP networks can take a long time [21] because most of the IP routing protocols are not designed to minimize network outages. In [21] it says that a single link failure can cause users to experience network service outages of several minutes even when the underlying network is highly redundant with plenty of spare bandwidth available and with multiple ways to route around the failure. Needless to say, depending on the application, outages of several minutes are not acceptable, for example, for e-commerce, voice over IP (VoIP) services, or transportation systems. Therefore, it is necessary to study ways to minimize the impact of link failures. For more information regarding the impacts of link failure, I would like to refer readers to [19, 20].

Depending on the type of the network, nodes have different names [1]. For example, routers, switches, servers etc. can be called nodes depending on the network type. Suppose that a source server communicate to a destination server and the destination server is overloaded. The destination server will queue the request to process later, making the response time longer. On the other hand, a relatively free destination server that can serve the source server's request may respond without any delay. In distributed computing environments, node load becomes a key challenge in node selection [22]. Zhang et al. [23] presented an scheme that prevents node overload by delaying routing updates. Chang et al. [24] investigated the router performance of commercial routers depending on the workload. They found that some routers exhibit performance degradations in overloaded routers. Reference [1] explains the effects of routing instability caused by overloaded nodes. It also explains the impacts of "Network storm effect", which is caused by overloaded nodes. In order to indicate that a router is in overloaded condition, a "overload bit" is used to communicate the overloaded situation to other routers. Sending excessive amount of state by one node to other nodes within the network, overloads the other nodes. This is considered a large threat to todays routing protocols [26]. Node overload can frequently lead to problems in queuing delay of data processing [2]. This delay caused by node overload can adversely affect the quality of service. Therefore, it is necessary to study ways to minimize the impact of node overload.

In recent years, the idea of robust optimization has gained much attention in the field of optimization [3, 4]. Robust optimization deals with uncertainty in the data of optimization problems. Under this framework, the objective and constraint functions are only assumed to belong to certain sets in function space (the so-called “uncertainty sets”). The goal is to make a decision that is feasible no matter what the constraints turn out to be, and optimal for the worst-case objective function (min-max objective). Mulvey et al. [5] presented an approach that integrates goal programming formulations with scenario-based description of the problem data. Soyster [6] et al. presented a linear optimization model to construct a solution that is feasible for all input data such that each uncertain input data can take any value from an interval. Robust optimization is being used in many real-world applications such as finance, mechanics, and control etc. due to the recent progress in linear programming.

1.2 Routing in IP networks

Routing is the main process used to deliver data in networks. IP networks use hop-by-hop routing model. This means that each router is only responsible for forwarding a datagram to another router. This process continues until the datagram reaches its destination or times-out because its path is too long. Random-walk routing techniques are not particularly reliable, and so it is important that the routers in a network have a coordinated approach to decide which is the next hop along the path to a destination. Routing techniques essentially pass information between neighboring routers and use this data to build the shortest paths to all destinations. These are then stored in a routing table and passed on to the router’s neighbors. The path computation model deployed in IP networks is iterative and distributed.

These next hop information in the routing table can either be set manually or dynamically. In simple networks, routing tables can be manually configured or learned from the configuration of interfaces on the router [1]. In more complex networks in which there are many routers arranged in a mesh with lots of links between routers, each link having different capabilities, manual configuration becomes onerous. More important - when a link or a router fails, all of the routing tables across the whole network must be updated to take account of the change. Similar changes are desirable when failures are repaired or when new links and nodes are added. Therefore, it is desirable to use a routing protocol to determine the next hop information dynamically.

Open Shortest Path First (OSPF) [28] is a widely used protocol that routes IP packets dynamically. It gathers link state information from available routers and constructs a topology map of the network. The topology determines the routing table presented to the Internet Layer which makes the routing decisions based solely on the destination IP address found in IP packets. OSPF detects changes in the topology, such as link failures, very quickly and converges on a new loop-free routing structure within seconds using detour routes. It computes the shortest path tree for each route using a method based on Dijkstra's algorithm [29], a shortest path first algorithm. In order to select a specific path in OSPF, there are costs assigned to all of the links. Traffic engineering is the process of predetermining the path through the network that various flows of data will travel and link cost (link weight) is used to introduce traffic engineering in OSPF routing model. Link weights must be calculated considering network topology, network resources etc. so that the calculated link weights would satisfy the requirements set by the network operator.

1.3 Link weight optimization

Based on weights of links in the network, routing protocols find the end-to-end path for each source-destination pair such that the sum of link weights on the path is minimized, so called the shortest path. Hence, to find a 'good' path for each source and destination pair that satisfies some network constraints such as the quality of service (QoS) of each session, energy consumption, and the target utilization of each link, it is important to assign the appropriate weight for each link in the network. A simple default weight setting policy suggested by Cisco [31] is to make the weight of a link inversely proportional to its capacity. But this simple weight setting policy do not emphasize on traffic matrix (traffic demand) or required traffic over link. So still this does not give the most optimal performance for the oblivious routing.

A more efficient way is optimizing link weights for a given topology considering the traffic over the network which will reduce a certain objective function.

This scheme is called as Start-time Optimization (SO). Optimization provides network operators with a powerful method for traffic engineering. Its general objective is to distribute traffic flows evenly across available network resources in order to avoid network congestion and quality of service (QoS) degradation. Multiple algorithms that focus on implementing SO are presented in [32, 33]. These studies all assume that the network topology and the traffic matrix are given. Unfortunately, SO is weak against link failures. When a link failure occurs, traffic related to that link should be re-routed through other active links, which can create an excess load to any other link and causes congestion in the network. Link failure is considered as a failure but if we consider it as a topology change in the network, we can take precautions for this problem in routing. When a link fails, the network takes a new shape. So we can take topology as a variable which can change with time because of link failure. For every possible link failure, we can get a new topology. If we consider a network topology as shown in Figure 1.1(a), all the new topologies created by single link failure will be shown as Figure 1.1(b).

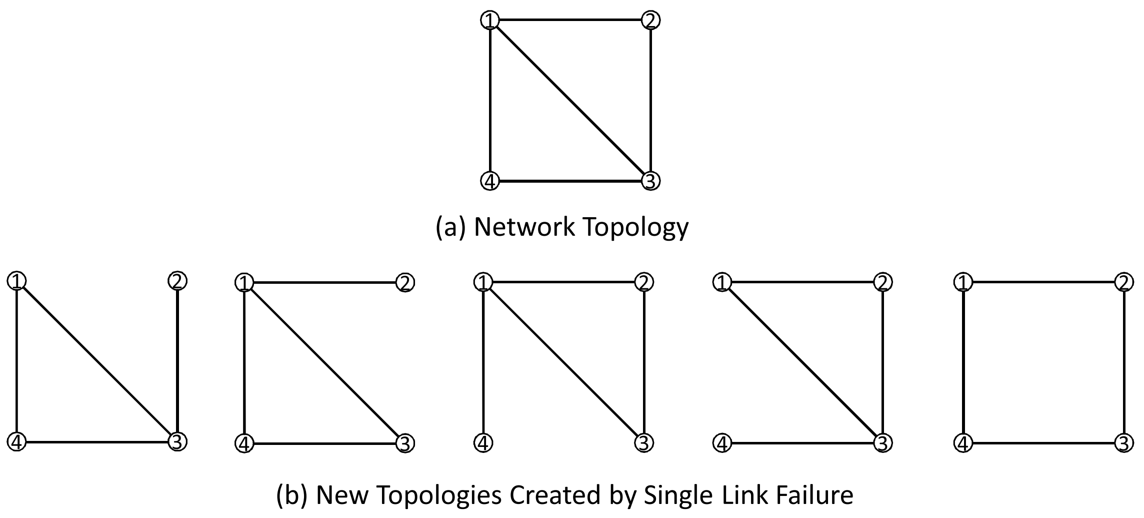


Figure 1.1: A simple network topology and topologies after single link failure

The weakness of SO can be overcome by computing a new optimal set of link weights whenever the topology is changed. It can be said that this approach, called Run-time Optimization (RO) provides the best routing performance after link failure. However, updating link weights in any case may lead to network instability [34] and 50% of link failures last less than one minute [17]. The analysis in [17] observed that more than 70% of transient failures that lasted less than 10 minutes are single link failures. Therefore, it seems reasonable to target the one-time configuration of link weights that can handle any single link failure.

PSO is a scheme that determines, at the start time, a suitable set of link weights that can handle any single link failure scenario preventively [35]. The objective of this scheme is to determine, at the start time, the most appropriate set of link weights that can avoid both unexpected network congestion and network instability, the drawbacks of SO and RO, regardless of which link fails. PSO considers all possible single link failure scenarios at start time in order to determine a suitable set of link weights. However, the current PSO scheme is usable when the traffic demand of the network is exactly known. This traffic model is called a pipe model. However, in general, the pipe model is proved to be very difficult for network operators to use [36, 37, 38].

1.4 Traffic demand models

In terms of bandwidth specification, networks can be divided into a pipe model and a hose model. The pipe model needs to specify the traffic demand between any two nodes. It means that the entire traffic demand (traffic matrix) of the network is given [36, 37, 38]. Figure 1.2 shows an example of the pipe model and its traffic demand. Even though the pipe model generally achieves a high performance, it is difficult to measure or predict the traffic matrix in reality [37]. The PSO scheme [35] introduced in section 1.3 is based on the pipe model. It calculates an optimal link weight set that can handle single link failures assuming the traffic matrix is given.

Since the prediction of the actual traffic requirement as required in the pipe model is difficult, it is, therefore, considered to be easier for network operators to specify the traffic as just the total ingress (α_i) and egress traffic (β_i) (i.e., the amount of traffic that can be sent to and received from the backbone network) at each node. The traffic model that has achieved this specification is called

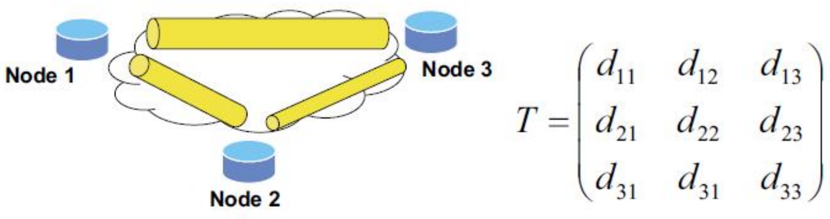

Node 1
Node 2
Node 3
$T = \begin{pmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{31} & d_{33} \end{pmatrix}$

Figure 1.2: Pipe model and its traffic demand

a hose model [40]. An example of the hose model is shown in Figure 1.3. The ingress and egress bandwidth requirement is as Equation (1.1). Chu et al. formulated the general routing problem of the hose model and presented an algorithm for solving the link weight searching problem in [39]. The hose model presents some challenging problems for traffic engineering as the hose model only needs to specify the total ingress bandwidth requirement and the total egress bandwidth requirement at each node.

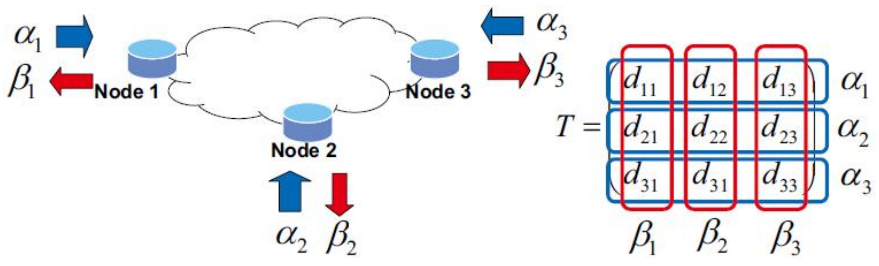

α_1
β_1
Node 1
Node 2
Node 3
α_2 β_2
α_3
β_3
$T = \begin{array}{ccc|c} d_{11} & d_{12} & d_{13} & \alpha_1 \\ d_{21} & d_{22} & d_{23} & \alpha_2 \\ d_{31} & d_{31} & d_{33} & \alpha_3 \\ \hline \beta_1 & \beta_2 & \beta_3 & \end{array}$

Figure 1.3: Hose model and its traffic constraints

$$\sum_j d_{ij} \leq \alpha_i \tag{1.1a}$$

$$\sum_i d_{ij} \leq \beta_j \tag{1.1b}$$

1.5 Apache Hadoop

Apache Hadoop (Hadoop) [41], an open-source implementation of Google's MapReduce [42] framework, is a parallel-distributed processing framework that allows organizations to efficiently process very large datasets using commodity hardware in a realistic time. Hadoop has become the most popular parallel-distributed framework for processing large-scale data because it hides the complexity of distributed computing, scheduling, and communication while providing fault-tolerance. In general, Hadoop clusters are made with tens to thousands of servers. Hadoop makes use of inexpensive, industry standard commodity servers to store and process data rather than relying on specially built proprietary servers. Thus, most of the fortune 500 companies exploit Hadoop to process large-scale data within a reasonable budget. Initially, Hadoop was used at large companies such as Yahoo, Facebook, eBay etc., who already had large amounts of data and their Hadoop clusters are usually deployed in on-premise physical clusters.

HDFS (Hadoop Distributed File System) is the distributed storage layer that is responsible for storing data in Hadoop. Data in HDFS are broken into blocks and stored as replicas in multiple (at least three), different servers for fault-tolerance and availability purposes. Data processing tasks in Hadoop such as MapReduce jobs, Hive queries or Spark jobs access data stored in HDFS as required by the task. HDFS is designed with write-once-read-many access model [43, 44] to attain high throughput of data access. Write-once-read-many means that once data is written to HDFS, that particular data will be read by processing tasks many times over the time. Therefore, improving how data is read in HDFS has a larger impact on the overall performance of HDFS compared to improving how data is written.

Let us consider how HDFS fetches (this thesis uses fetch in similar meaning to read since a client that wants to read data has to fetch data from a remote server) a particular data block in a high level. As we described earlier, there are three replicas of the same data block in three different servers. Therefore, HDFS has the freedom to select one of those servers when reading a particular data block. As shown in Figure 1.4, selecting one server is similar to selecting a path between two servers; if Server #1 is selected it is similar to selecting Path 1, if Server #2 is selected it is similar to selecting Path 2, and if Server #3 is selected it is similar to selecting Path 3. When there are three choices to select from, how HDFS chooses one among them is an interesting question. When fetching a data block, HDFS finds the best server to fetch data from by only comparing the network distance of servers that hold a replica of the data block. The concept called “rack awareness” is used to calculate the network distance. We will explain “rack awareness” and how network distance is calculated in section 4.1 in detail. As we explained in section 1.1, server load is a major factor that should be considered when path selection decisions are made. However, HDFS does not consider the server load when selecting a server to fetch data from.

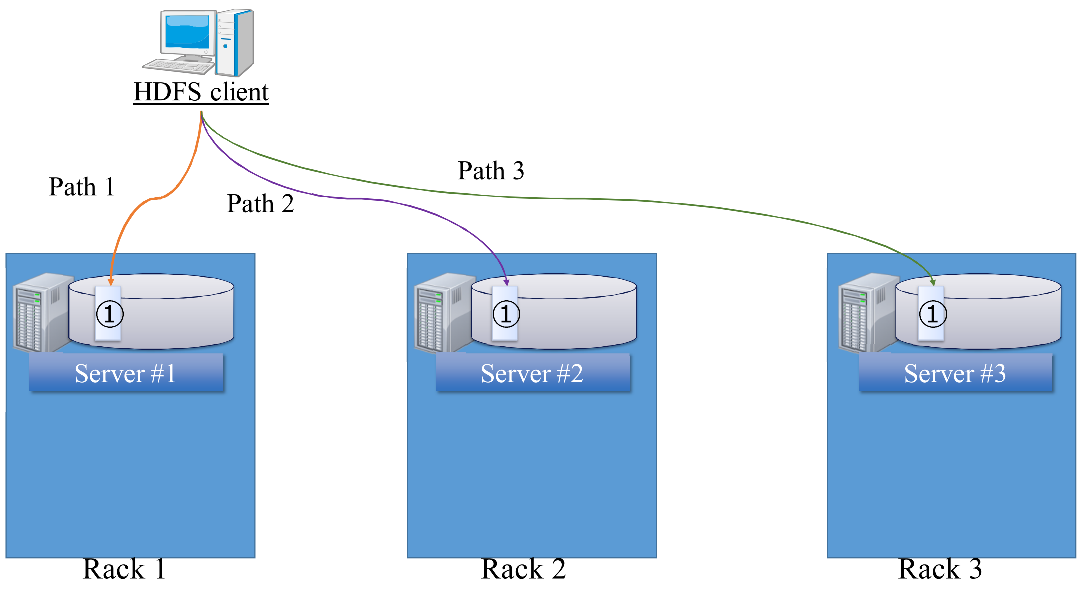


Figure 1.4: Replica selection is equal to path selection

There are multiple studies in literature that studied on improving resource management, job scheduling, and replica placement in Hadoop. Tan et al. studied analytical models for scheduling jobs based on extensive measurements and source code investigations [48]. Rasley et al. studied queue management techniques, such as appropriate queue sizing, prioritization of task execution via queue reordering, starvation freedom, and careful placement of tasks to queues [49]. Li et al. studied novel predictive scheduling framework to enable fast and distributed stream data processing [50]. Zaharia et al. addressed the problem of data locality while keeping fairness during task execution [51]. Many improvements for enhancing data placement (data write) in HDFS were studied in [70, 71, 72].

1.6 Problem formulation

1.6.1 Path selection scheme considering traffic demand and link failure uncertainty

OSPF is a widely used routing protocol in IP networks. OSPF requires link weights to be assigned to each link to calculate the shortest paths between every source-destination pair. In order to provide efficient routing to the users, these link weights must be calculated according to the network topology and service provider's requirements. SO calculates an optimal set of link weights for a given topology when the exact traffic demand is known (pipe model) that will reduce a certain objective function, such as congestion ratio or energy consumption of the network. However, apart form the difficulty of exactly knowing the traffic demand, SO is weak against link failures and traffic demand fluctuations because it only considers one topology and one traffic pattern at a time. A scheme applying the hose model, that can handle traffic demand fluctuations, on the other hand, provides robustness against traffic demand uncertainty. Chu et al. studied how to apply SO to the hose model. PSO on the other hand can calculate an optimal link weight set which is robust against link failures at start time if the traffic demand is exactly known. However, how to apply PSO for the hose model has not been studied. Being able to handle traffic demand fluctuations while guaranteeing robustness against link failure is desirable for network operators. This is the objective of the first part of this thesis.

1.6.2 Path selection scheme considering node load uncertainty

Hadoop is an open-source parallel-distributed processing framework. Hadoop can efficiently use the resources of commodity servers that are connected to a network. The storage layer of Hadoop, called HDFS, stores data as blocks and each block is replicated to multiple servers for fault-tolerance and availability purposes. HDFS is designed with write-once-read-many access model meaning every improvement made to data reading mechanism contributes to improving overall performance of the cluster. When a client reads a data block, it has the choice of selecting one server among the three servers that holds the same data block as replicas. HDFS selects a server to fetch data from considering the network distance and do not consider the server load of each server even though it affects the cluster performance. Therefore, robustness against server load in HDFS is desirable for Hadoop clusters. This is the objective of the second part of this thesis.

1.7 Contributions

This thesis proposes robust path selection schemes under uncertain network conditions. It presents a path selection scheme considering traffic demand and link failure uncertainty, and a path selection scheme considering server load uncertainty.

In the first part, path selection problem considering link failure under uncertain traffic conditions is studied. Robust optimization approach presented in this part is based on the hose model. In the hose model, contrary to the pipe model, the exact traffic matrix does not need to be specified by the network operators. The network operators can however set bounds on the total ingress and egress traffic at each node from their experience. Modeling the traffic for hose model allows an optimization scheme to accommodate any traffic matrix that fits within the bounds at each node. Link failures in a network can be considered as a topology change as we explained in section 1.3. We introduce a link weight optimization model for OSPF that takes traffic matrix and topology as variables.

Usually, routing decisions are made to satisfy network operator's objectives such as reducing the network congestion, reducing the energy consumption, satisfying the target link utilization of each link etc. We formulate mixed integer linear programming (MILP) problems to reduce worst-case network congestion and to reduce worst-case energy consumption by putting unnecessary links into sleep mode. Due to the complexity of solving the MILP formulations, we present heuristics to calculate a link weight set that achieves satisfactory performance in practical time. The heuristics consider the worst-case traffic matrix and worst-case link failure topology to calculate a link weight set in practical time. Simulation results observe that the presented schemes, while being robust to link failure and traffic demand fluctuation, is able to reduce the worst-case network congestion and worst-case energy consumption.

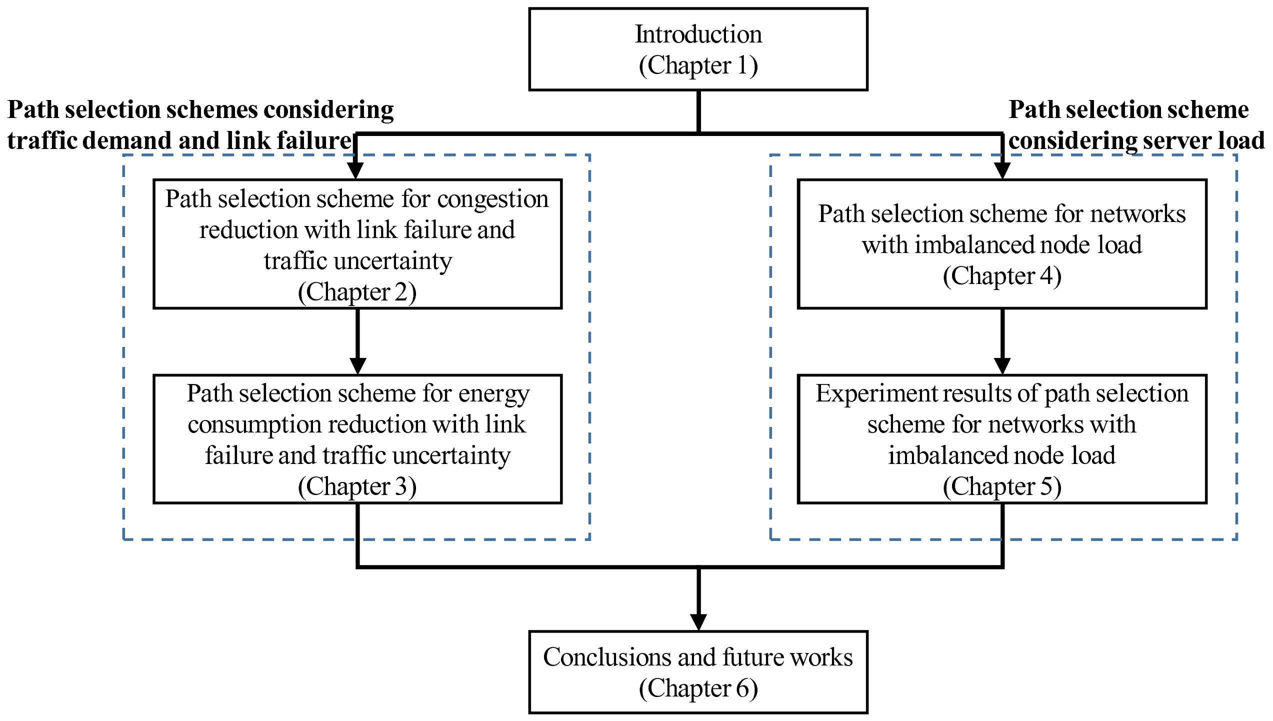
In the second part, a robust path selection problem considering node load is studied. In HDFS, rack information is used to calculate the network distance between servers notwithstanding the server load. This thesis presents a path selection scheme based on the delay distribution between servers for Hadoop clusters. This scheme calculates the round-trip time between all server pairs periodically. The presented scheme selects a server comparing the delay distribution between server pairs. In order to achieve this, changes are made to source code of HDFS. Using the changed source code, we rebuilt Hadoop software packages and configured multiple Hadoop clusters to do experiments. The experiment results observe that the presented scheme, while being robust to server load, is able to dynamically select the best server resulting shorter data fetch time compared to conventional Hadoop.

1.8 Organization of the thesis

The organization of the thesis is shown in Figure 1.5. Path selection schemes considering traffic demand and link failure uncertainty is presented in chapter 2 to 3. Chapter 4 to 5 present path selection scheme considering server load uncertainty. Chapter 6 concludes the thesis.

A scheme to calculate a link weight set that is robust against link failure and traffic demand fluctuation, minimizing the worst-case network congestion is presented in chapter 2. Chapter 2 also includes the MILP formulation and simulation results. A scheme to calculate a link weight set that is robust against link failure and traffic demand fluctuation minimizing the worst-case network energy consumption is presented in chapter 3. It also includes the MILP formulation and simulation results.

A scheme to select paths considering server load is presented in chapter 4. Since experimental results have more impact than simulation results, experimental environment information and results are introduced in chapter 5.



Introduction (Chapter 1)
Path selection schemes considering traffic demand and link failure
Path selection scheme considering server load
Path selection scheme for congestion reduction with link failure and traffic uncertainty (Chapter 2)
Path selection scheme for energy consumption reduction with link failure and traffic uncertainty (Chapter 3)
Path selection scheme for networks with imbalanced node load (Chapter 4)
Experiment results of path selection scheme for networks with imbalanced node load (Chapter 5)
Conclusions and future works (Chapter 6)

Figure 1.5: Organization of the thesis

Chapter 2

Path selection scheme for congestion reduction with link failure and traffic uncertainty

Traffic engineering is the process of predetermining the paths through the network that various flow of data will travel. Link weight is used to introduce traffic engineering in OSPF routing model. A link weight calculation scheme that is optimized to achieve the objective(s) of network service providers is required. In this chapter, a path selection scheme that is robust against link failure and traffic demand fluctuation with the objective of reducing the worst-case congestion is presented. In section 2.1, network congestion and the importance of reducing network congestion is introduced. Terminologies used in the later sections are defined in section 2.2. In section 2.3, the MILP formulation to reduce the worst-case congestion ratio is introduced and the heuristic algorithm is introduced in section 2.4. Section 2.5 presents the simulation results.

2.1 Network congestion

Network congestion is the result of a route in the network being heavily utilized. Network congestion can deteriorate network service quality, resulting in queuing delay, data packet loss and the blocking of new connections [56]. Since, in OSPF, routes are determined by link weights, link weights can be optimized so that each route in the network maintains a manageable congestion level.

One useful approach to enhance routing performance is to minimize the maximum link utilization rate, network congestion ratio, of all links in the network. Reducing the congestion ratio in the network is a common objective of network service providers [1]. Also, they must be prepared for the worst-case scenario, worst-case congestion ratio, to maintain a satisfying service. Therefore, network service providers configure routes in the network to minimize the worst-case congestion ratio [57] increasing the admissible traffic [56].

2.2 Terminologies

The network is described as a directed graph $G(V, E)$, where V is the set of nodes and E is the set of links. $v \in V$, where $v = 1, 2, \ldots, N$, indicates an individual node and a link from node $i \in V$ to node $j \in V$ is denoted as $(i, j) \in E$. N is the number of nodes and L is the number of links in the network, or $L = |E|$. c_{ij} is the capacity of link $(i, j) \in E$. The traffic volume on link $(i, j) \in E$ is denoted as u_{ij}. Since the probability of concurrent multiple link failures is much less than that of single link failures [17, 18], only single link failures in the network are considered in this study. F is the set of link failure indices l, where $l = 0, 1, 2, \cdots, L$ and $F = E \cup \{0\}$. The number of elements in F is $|F| = L + 1$. $l = 0$ indicates no link failure and $l(\neq 0)$ indicates the failure of link $(i, j) = l(\neq 0)$. The network in which lth ($\neq$0) link failed is described as a directed graph $G_l(V, E_l)$. As $G_0(V, E_0)$ indicates no link failure, $G_0(V, E_0) = G(V, E)$ and let $\tilde{G}$ be the set of network topologies. c^l_{ij} is the capacity of $(i, j) \in E_l$. If $(i, j) = l$, $c^l_{ij} = 0$. $W = \{w_{ij}\}$ is the link weight matrix of network G, where w_{ij} is the weight of link (i, j). Let $\{1, \ldots, w_{max}\}$ be the set of possible link weights. x^{pq}_{ij} is the portion of traffic from node $p \in V$ to node $q \in V$ routed through $(i, j) \in E$. $x^{pq}_{ij}(W, l)$ is used to represent the load distribution variables under link weights set W and link failure on node l.

Since an OSPF-based network uses the shortest path routing, load distribution $x^{pq}_{ij}(W, l)$ and routing are determined if link weights are known. In this study, it is assumed that equal-cost multi-path (ECMP) routing is employed, where traffic is evenly split among equal-cost paths [66]. Let a_p, b_p represent the maximum amount of ingress and egress traffic allowed to enter and leave the network at node p, respectively. Given the ingress and egress traffic constraints specified by $H = [(a_1, b_1), \ldots, (a_n, b_n)]$, there are many traffic matrices that satisfy the constraints imposed by H. A traffic matrix $T = \{d_{pq}\}$, where d_{pq} represents the traffic rate from node p to node q, is called a *valid* traffic matrix if it does not violate the constraints imposed by H. Let $\tilde{D}$ be the set of all valid Ts.

The network congestion ratio r refers to the maximum value of all link utilization ratios in the network. r is defined by,

$$r = \max_{(i,j)\in E} \frac{u_{ij}}{c_{ij}}, \tag{2.1}$$

where $0 \leq r \leq 1$.

Let ψ^{pq}be the length of the shortest path from p to q, δ_q^{ij} be a set of binary variables such that $\delta_q^{ij} = 1$ if link (i,j) is on a shortest path to node q, and 0 otherwise. $\pi^{ij}(p)$ and $\lambda^{ij}(p)$ are introduced variables to solve the dual explained in section 2.3. Let

$$f_{pq}^{i} = \frac{\text{the portion of traffic from } p \text{ to } q \text{ that arrives at } i}{m} \tag{2.2}$$

where m is the number of outgoing links incident from i that are on the shortest paths to node q.

The target of this chapter is to find the most appropriate set of link weights, W_{min}, for network G that minimizes the worst-case congestion ratio over link failure index $l \in F$ and traffic matrices $T \in \tilde{D}$. W_{min} is defined by,

$$W_{min} = \arg\min_{W\in w} \max_{G_l\in\tilde{G}} \max_{T\in\tilde{D}} r(G_l, T, W). \tag{2.3}$$

The traffic matrix $T \in \tilde{D}$ that maximizes the congestion ratio against all the single link failure scenarios of $G_l \in \tilde{G}$ is searched followed by the finding of the link weight set that minimizes the worst-case congestion ratio.

2.3 MILP formulation for congestion reduction

In the hose model the traffic is specified as only the total outgoing/incoming traffic (a_p, b_p) from/to node p is expressed as,

$$a_p = \sum_{q \in V} d_{pq}, \quad b_p = \sum_{p \in V} d_{pq}. \tag{2.4}$$

Since the traffic matrix can vary within a_p and b_p, one way to deal with the hose model is to consider the worst-case scenario [39]. In this case, using the given routing paths, the worst-case traffic matrices are generated under the hose boundary, a_p and b_p.

The problem formulation for generating the worst-case traffic matrices is as follows.

$$\max \quad \sum_{pq} x_{ij}^{pq}(W) d_{pq} \tag{2.5a}$$

$$\text{s.t.} \quad \sum_{q \in V} d_{pq} \leq a_p, \forall p \in V \tag{2.5b}$$

$$\sum_{p \in V} d_{pq} \leq b_q, \forall q \in V \tag{2.5c}$$

$$d_{pq} \geq 0, \forall p, q \in V \tag{2.5d}$$

With the constraints (2.5b)-(2.5c) a solution to this problem covers the hose model worst-case scenario. Also, since this problem has the same optimal solution as its dual problem, it can be replaced by the latter formulated in Equation (2.6). $\pi^{ij}(p)$ and $\lambda^{ij}(p)$ are introduced variables to replace the infinite number of variables d_{pq} in Equation (2.5) [39].

$$\min \quad \sum_{p \in V} a_p \pi^{ij}(p) + \sum_{p \in V} b_p \lambda^{ij}(p) \tag{2.6a}$$

$$\text{s.t.} \quad x_{ij}^{pq}(w) \leq \pi^{ij}(p) + \lambda^{ij}(q), \quad \forall p, q \in V, (i,j) \in E \tag{2.6b}$$

$$\pi^{ij}(p), \lambda^{ij}(p) \geq 0, \quad \forall p \in V, (i,j) \in E \tag{2.6c}$$

Using the dual formulation we can derive the mixed-integer programming formulation of network optimization for minimizing the worst-case congestion ratio, r_{ij}, in case of link failure under the hose model traffic as shown in Equation (2.7).

$$\min \quad r \tag{2.7a}$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in E_l} x_{ij}^{pq}(W,l) - \sum_{j:(j,i)\in E_l} x_{ji}^{pq}(W,l) = 1, \quad \forall p,q \in V, i = p, l \in F \tag{2.7b}$$

$$\sum_{j:(i,j)\in E_l} x_{ij}^{pq}(W,l) - \sum_{j:(j,i)\in E_l} x_{ji}^{pq}(W,l) = 0, \quad \forall p,q \in V, i(\neq p,q), l \in F \tag{2.7c}$$

$$\sum_{p\in V} a_p \pi^{ij}(p) + \sum_{p\in V} b_p \lambda^{ij}(p) \leq r c_{ij}^{l}, \quad \forall (i,j) \in E_l, l \in F \tag{2.7d}$$

$$x_{ij}^{pq}(W,l) \leq \pi^{ij}(p) + \lambda^{ij}(q), \quad \forall p,q \in V, (i,j) \in E_l, l \in F \tag{2.7e}$$

$$0 \leq f_{pq}^{i}(l) - x_{ij}^{pq}(W,l) \leq 1 - \delta_q^{ij}(l), \quad \forall p,q \in V, (i,j) \in E_l, l \in F \tag{2.7f}$$

$$x_{ij}^{pq}(W,l) \leq \delta_q^{ij}(l), \quad \forall p,q \in V, (i,j) \in E_l, l \in F \tag{2.7g}$$

$$0 \leq \psi^{jq}(l) + w_{ij} - \psi^{iq}(l) \leq (1 - \delta_q^{ij}(l))U, \quad \forall q \in V, (i,j) \in E_l, l \in F \tag{2.7h}$$

$$1 - \delta_q^{ij}(l) \leq \psi^{jq}(l) + w_{ij} \leq \psi^{iq}(l), \quad \forall q \in V, (i,j) \in E_l, l \in F \tag{2.7i}$$

$$\pi^{ij}(p), \lambda^{ij}(p) \geq 0, \qquad \forall p \in V, (i,j) \in E \tag{2.7j}$$

$$f_{pq}^{i}(l) \geq 0, \qquad \forall p,q,i \in V, l \in F \tag{2.7k}$$

$$\delta_q^{ij}(l) \in \{0,1\}, \quad \forall q \in V, (i,j) \in E_l, l \in F \tag{2.7l}$$

$$1 \leq w_{ij} \leq w_{max}, \qquad \forall (i,j) \in E_l, l \in F \tag{2.7m}$$

The objective of the above mixed-integer programming formulation is to find the optimal set of link weights in order to minimize the worst-case congestion ratio under single link failure. Equations (2.7b)-(2.7c) represent the traffic flow constraints. For a given set of hose model traffic, the worst-case scenario is considered as in fixed routing; we maximize the traffic flow over the active link with constraints (2.7d)-(2.7e) and (2.7j). Constraints (2.7f) and (2.7g) are the flow splitting constraints such that traffic is split to the shortest paths according to the even distribution rule. Constraints (2.7h) and (2.7i) are the shortest path constraints. If link (i, j) does not lie on any shortest path to node q (i.e., $\delta_q^{ij}(l) = 0$), $\psi^{jq}(l)+w_{ij}-\psi^{iq}(l) \geq 1$ must hold because $w_{ij} \geq 1$. This is stated by Equation (2.7i). On the other hand, constraint (2.7h) implies that $\psi^{jq}(l)+w_{ij}-\psi^{iq}(l) = 0$ if link (i, j) is on one of the shortest paths to node q. In addition, when $\delta_q^{ij}(l) = 0$, Equation (2.7h) becomes redundant if U (an artificial constant) is sufficiently large [39]. Constraints (2.7j)-(2.7m) provide the ranges for the variables.

Unfortunately, this mixed-integer program is NP-Hard and limits its applicability to only small networks. Therefore, in the following section, we present a heuristic approach to optimize the link weights in case of link failure under the hose model traffic. The presented heuristic approach has been found to yield good performance on all the sample networks tested in this study.

2.4 Heuristic approach

The heuristic scheme considers the worst-case traffic matrix and one link failure topology to decrease the congestion ratio by changing link weights. It uses tabu search [60] and an efficient objective function in the optimization process to reduce the computation time. The presented heuristic approach is divided into three stages.

At Stage 1, we generate the traffic matrices that lead to the maximum load on each link $(i, j) \in E$ in the allowable traffic bound (a_p, b_p).

At Stage 2, we calculate the congestion ratios for all the traffic matrices against single link failure and find which link failure topology gives the maximum congestion ratio. Within all of the traffic matrices against single link failure topologies, the traffic matrix that maximizes the congestion ratio is chosen. Then we try to reduce that congestion ratio by changing the link weight of the most congested link. We continue updating the link weights until all the link weights reach the maximum link weight.

At Stage 3, the improvement of the new link weight set is evaluated. If the link weight set is accepted, the algorithm terminates. If not, it returns to Stage 1.

The description of the presented heuristic approach is as follows.

Stage 1: Generating traffic matrices

- Step 1: *Set initial link weights*
  At first, the link weights are generated randomly. Once link weights are known, the shortest paths and routing $x_{ij}^{p}q(W)$ are determined.

- Step 2: *Generate traffic matrices*
  For each link (i, j), the linear programming formulation in Equation (2.5) is used to find the worst-case traffic matrix T^{ij} that leads to the maximum load appeared on link (i, j).

The traffic matrix T^{ij} that achieves the maximum link utilization for each link (i, j) will be added to the set $\tilde{D}$ if it is not in $\tilde{D}$ already.

Stage 2: Searching for an optimal link weight set
The updated set $\tilde{D}$ produced at Stage 1 is used to search for new link weights that reduce an objective function. The objective function considerably affects the efficiency of the algorithm. Let $\tilde{r}$ denote the congestion ratio for set $\tilde{D}$. Let r_T be the maximum link utilization ratio for a specific traffic matrix T. Therefore, $\tilde{r} = \max_{T \in \tilde{D}} \{r_T\}$. Although our goal is to minimize $\tilde{r}$, we find that $\tilde{r}$ is not a suitable objective function in the optimization process because changing a link weight reduces one r_T but also often increases a different $r_{T'}$. This means that the improvement of $\tilde{r}$ cannot be done in any iteration.

A better objective function, as used in [39], should include r_T for all traffic matrices in $\tilde{D}$. The sum of individual cost function $\phi(r_T)$ of r_T is chosen as the objective function $F(\tilde{D})$ of the presented scheme. $F(\tilde{D})$ is defined as,

$$F(\tilde{D}, G_l)_{|\text{for given weights}} = \max_{G_l \in \tilde{G}} \sum_{T \in \tilde{D}} \phi(r_T), \tag{2.8}$$

where $\phi(r_T)$ increases with r_T. Inspired by [39], we adopt the following convex piecewise linear cost function for $\phi(r_T)$.

$$\phi(r_T) = \begin{cases} r_T, & 0 \le r_T < \frac{1}{3} \\ 3r_T - \frac{2}{3}, & \frac{1}{3} \le r_T < \frac{2}{3} \\ 10r_T - \frac{16}{3}, & \frac{2}{3} \le r_T < \frac{9}{10} \\ 70r_T - \frac{178}{3}, & \frac{9}{10} \le r_T < 1 \\ 500r_T - \frac{1468}{3}, & 1 \le r_T < \frac{11}{10} \\ 5000r_T - \frac{16318}{3}, & \frac{11}{10} \le r_T < \infty. \end{cases} \tag{2.9}$$

In [39], it is stated that they have tried different convex objective functions and they all have similar performances in terms of network congestion ratio minimization. Thus, the presented scheme also uses the same convex objective function.

- Step 1: *Initialize*
  Variable F_{min}, which is used to store the value of the objective function, is set to infinite. The repetition counter I_c, which is used to stop the oscillation of the objective function, is also set to zero.

- Step 2: *Choose a traffic matrix*
  At first the repetition counter I_c is checked. If it is greater than the allowed repetition number, go to Step 1 of Stage 3. If not, the traffic matrix T_{max} that maximizes the cost function defined in Equation 2.9 against all the single link failure instances is selected.

- Step 3: *Find the most congested link*
  By using the traffic matrix T_{max}, which was selected in Step 2 of Stage 2, the most congested link, $(i, j)_{cong}$, in the network against single link failures is selected.

- Step 4: *Update the link weight*
  The link weight of the most congested link, selected in the previous step is increased by the minimum value that changes at least one route passing through the link for all single link failure scenarios. Therefore, the congestion of the most congested link is decreased. The updated link weight set is inserted into the tabu list. If the updated link weight exceeds the upper limit of the feasible link weight, w_{max}, go to Step 1 of Stage 3.

- Step 5: *Evaluate the objective function*
  For the updated traffic distribution obtained in Step 4 of Stage 2, the objective function of Equation 2.8 is calculated and compared with that of the old weight set. If the value of Equation 2.8 for the new weight set is greater than that of the old weight set, repetition counter I_c is reset to zero and the new weight set is set as W_{min} and go to Step 2 of Stage 2. Otherwise, repetition counter I_c is increased by one and go to Step 2 of Stage 2.

Stage 3: Choosing an optimal link weight set

- Step 1: The congestion ratio r for W_{min} is calculated and, if r differs from $\tilde{r}$ by a predefined ϵ, the algorithm terminates. If not, go to Step 2 of Stage 1 and start from the calculation of traffic matrices. W_{min} is an optimal link weight set for the given network against single link failure under traffic demand fluctuations.

Since the traffic matrices play an important role in the effectiveness of the presented scheme, we randomly use a significantly large number of independent initial link weight sets (different initial weight sets may give different worst-case congestion ratios). The link weight set that gives the minimum congestion ratio against single link failure is selected as an optimal link weight set.

2.5 Simulation results

The simulation environment that we used are described as follows. In order to compare the results of MILP formulation and heuristic algorithm, four relatively small scale sample networks are used as shown in networks 1-4 of Figure 2.1. To determine the basic characteristics of the presented heuristic approach, eight sample networks are used as shown in networks 5-12 of Figure 2.1. Networks 5-11 mirror the typical backbone networks used to evaluate the routing performance in [61]. Network 12 is a random network generated using the BRITE topology generator [62] and the *Waxman's Probability model* was used to create it. Table 2.1 summaries the basic characteristics of the sample networks in Figure 2.1. The link capacities of the sample networks were randomly generated with uniform

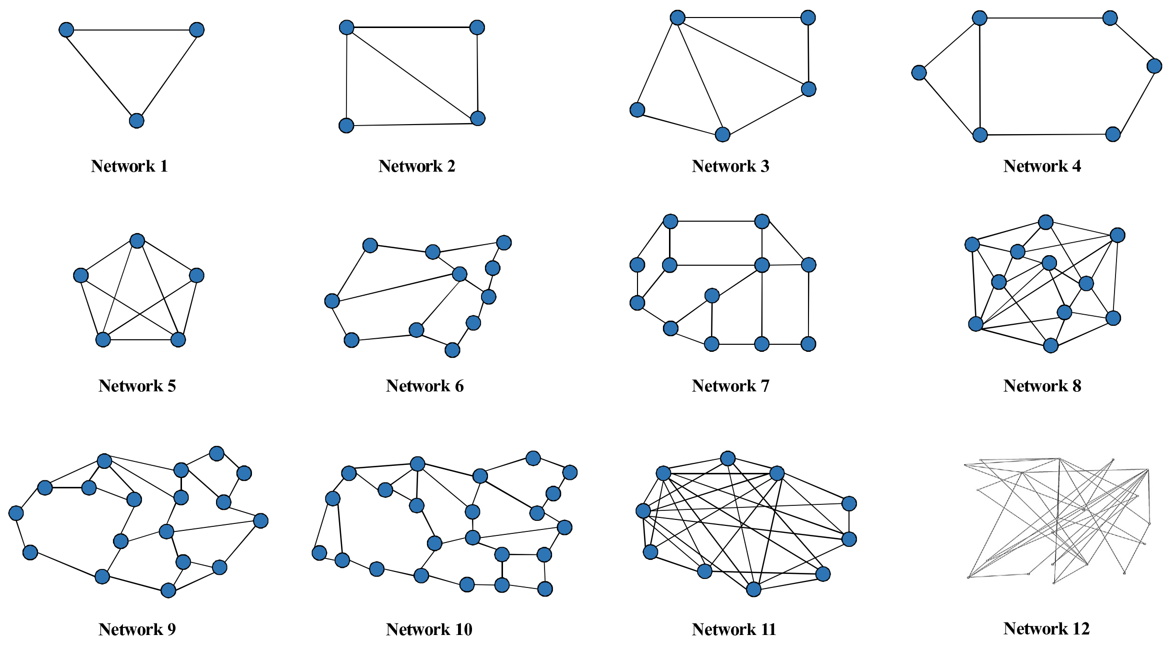


Figure 2.1: Sample networks used in the simulations

distribution in the range of ($10U_c$, $100U_c$), where U_c[Gbit/s] is given a constant integer value. The maximum link weight, w_{max}, is set at 100. We confirmed that $w_{max} > 100$ provides the same results as $w_{max} = 100$ in our examined networks. I_c is set to 1000 after observing the trend of the worst-case congestion reduction of the presented scheme. The simulation program is coded in C language and executed on a Linux computer with 20GB of RAM. The linear programming problem in Equation (2.5) is solved using the IBM ILOG CPLEX Optimization Studio 12.4.

First, the performance of the presented heuristic approach is compared with the MILP formulation, which gives the optimal link weight set.

Let R denote the worst-case network congestion ratios of the sample networks 1-4 shown in Figure 2.1. The normalized worst-case network congestion ratio of the MILP is denoted as R_{MILP} and the normalized worst-case network congestion ratio of the presented heuristic approach is denoted as R_{PSO-C}. The worst-case network congestion ratios are normalized using the results of MILP. The calculated worst-case network congestion ratios are shown in Table 2.2 and following

Table 2.1: Characteristics of the sample networks used to compare the heuristic and conventional schemes

Network	No. of nodes	Average node degree	No. of links (bidirectional)
1	3	2.00	3
2	4	2.50	5
3	5	2.80	7
4	6	2.33	7
5	5	3.60	9
6	11	2.54	14
7	12	3.00	18
8	11	4.73	26
9	18	3.00	27
10	23	2.78	32
11	10	5.60	28
12	20	3.70	37

relationship is observed from the results.

$$R_{MILP} = R_{PSO-C} \tag{2.10}$$

This indicates that the link weight set calculated using the heuristic approach is able to provide similar performance as compared to the MILP for relatively small scale networks. This is related to the fact that the paths between node pairs in small scale networks are substantially fixed. Therefore, the path selected by both MILP and the heuristic are the same. Network 4 is the largest network we are able to solve due to computation complexity of the MILP formulation.

Then, the performance of the presented heuristic approach is compared with that of SO and RO via simulations. Network congestion ratio r is the performance measure of the evaluation.

The congestion ratio of SO without any link failure is used to normalize the calculated network congestion ratios of the sample networks. Let $r(l)$ denote the network congestion ratio for link failure index $l \in F$. The normalized network

Table 2.2: Comparison of worst-case network congestion ratios of MILP and the heuristic

Network	R_{MILP}	R_{PSO-C}
1	1.00	1.00
2	1.00	1.00
3	1.00	1.00
4	1.00	1.00

congestion ratio of SO is denoted as $r_{SO}(l)$, the normalized congestion ratio of RO is denoted as $r_{RO}(l)$, and the normalized congestion ratio of the presented approach, $PSO - C$ is denoted as $r_{PSO-C}(l)$.

The worst-case network congestion ratios, $\max\limits_{l\in F} r_{SO}(l)$, $\max\limits_{l\in F} r_{RO}(l)$, and $\max\limits_{l\in F} r_{PSO-C}(l)$ of the sample networks 5-12 presented in Figure 2.1 for all single link failure scenarios are calculated as shown in Table 2.3. For the worst-case network congestion ratio for single link failure, the following relationship is observed.

$$\max_{l\in F} r_{RO}(l) \leq \max_{l\in F} r_{PSO-C}(l) \leq \max_{l\in F} r_{SO}(l) \tag{2.11}$$

This indicates that the presented scheme is able to reduce the worst-case network congestion ratio as compared with SO. It also avoids the run-time link weight changes, which would cause network instability. As expected RO gives the optimal performance when a link failure occurs even though RO may lead to network instability. The achieved reduction rate of the worst-case congestion ratio, α, is defined as,

$$\alpha = \frac{\max\limits_{l\in F} r_{SO}(l) - \max\limits_{l\in F} r_{PSO-C}(l)}{\max\limits_{l\in F} r_{SO}(l)}. \tag{2.12}$$

α is also shown in Table 2.3.

The normalized congestion ratios of no link failure are shown in Table 2.4. For the case of no link failure,

$$r_{RO}(0) = r_{SO}(0) \leq r_{PSO-C}(0) \tag{2.13}$$

Table 2.3: Comparison of worst-case network congestion ratios of the heuristic and conventional schemes for single link failure scenarios

Network	$\max_{l \in F} r_{SO}(l)$	$\max_{l \in F} r_{RO}(l)$	$\max_{l \in F} r_{PSO-C}(l)$	α
5	1.68	1.32	1.36	0.19
6	1.45	1.14	1.20	0.17
7	1.81	1.54	1.54	0.15
8	2.09	1.57	1.62	0.23
9	1.65	1.12	1.17	0.29
10	2.88	2.13	2.20	0.21
11	1.94	1.56	1.56	0.20
12	3.00	1.90	2.10	0.30

is observed. When there is no link failure, the congestion ratio of the link weight set obtained using PSO-C may be higher than that of SO or RO. This is because the objective of PSO-C is to reduce the worst-case network congestion ratio when link failure occurs. β is the deviation between $r_{PSO-C}(0)$ and $r_{SO}(0)$. β is defined as,

$$\beta = \frac{r_{PSO-C}(0) - r_{SO}(0)}{r_{SO}(0)}. \tag{2.14}$$

β is also shown in Table 2.4. When there is no link failure, β is the "penalty" PSO-C has to "pay" to reduce the worst-case network congestion. The non-normalized simulation results show that the worst-case congestion ratios under single link failure is significantly larger compared to no link failure scenario.

The worst-case congestion ratios of several random networks are calculated to understand the relationship between the worst-case congestion ratio and network topology. These random networks are generated using the BRITE [62] internet topology generator by changing the number of nodes N and the number of adjacency nodes m of the network. The Waxman's probability model is used for interconnecting the nodes of the topology, which is given by,

$$P(u, v) = A \ \exp(-\frac{d}{BL}), \tag{2.15}$$

Table 2.4: Comparison of network congestion ratios with no link failure

Network	$r_{SO}(0)(= r_{RO}(0))$	$r_{PSO-C}(0)$	β
5	1.00	1.03	0.03
6	1.00	1.15	0.15
7	1.00	1.07	0.07
8	1.00	1.12	0.12
9	1.00	1.05	0.05
10	1.00	1.10	0.10
11	1.00	1.04	0.04
12	1.00	1.17	0.17

where, $0 < A$, $B \leq 1$, d is the Euclidean distance from node u to node v, and L is the maximum distance between any two nodes. A and B are set to 0.15 and 0.2, respectively. The number of nodes, N, is set to $8, 10, 12, 14, 15$ and the number of adjacency nodes m is set to $3, 4, 5, 6$. The characteristics of the generated random network topologies are shown in Table 2.5 and the networks are shown in Figure 2.2.

The dependency of α on N and m is shown in Figure 2.3. This result shows that α is increasing with N for higher values of m ($m = 5, 6$). It means the difference between the worst-case congestion ratios of SO and the presented scheme is increasing. This may relate to the increase in routing flexibility as N and m become higher. On the other hand, α fluctuates for smaller values of m. Smaller values of m or smaller number of neighbor nodes limits the selection of paths between source-destination pairs. This limited number of paths restricts the flexibility of the routing, causing α to fluctuate when links fail.

Figure 2.4 shows the dependency of β against N and m. Figure 2.4 indicates that the presented scheme is able to achieve the same result as SO for no link failure when the network becomes larger. This may occur because there is a wider number of routes to chose from when the network is larger.

In order to further show the effectiveness of the presented scheme, the worst-case congestion ratios of the presented scheme are compared with those of the two following link weight setting schemes. One is a scheme in which a link weight

Table 2.5: Characteristics of the random sample networks

N	m	Average node degree	No. of links (bidirectional)
8	3	4.50	18
8	4	5.00	20
8	5	3.50	14
8	6	2.75	11
10	3	5.00	25
10	4	5.80	29
10	5	5.60	28
10	6	4.80	24
12	3	5.83	35
12	4	6.83	41
12	5	7.00	42
12	6	7.17	43
14	3	6.00	42
14	4	7.43	52
14	5	7.86	55
14	6	8.29	58
15	3	6.00	45
15	4	7.33	55
15	5	8.67	65
15	6	8.93	67

is inversely proportional to its capacity [31]. We call it the IPC scheme. The other scheme is the one in which all the link weights are set to one. As a result, minimum-hop routing is achieved. We call it the min-hop scheme. Table 2.6 shows the worst-case congestion ratios of the three schemes, which are normalized by that of the presented scheme. Table 2.6 indicates that the presented scheme reduces the worst-case congestion ratio, compared to the IPC scheme and the min-hop scheme. This is because the presented scheme determines link weights considering any single link failure so as to minimize the worst-case congestion ratio.

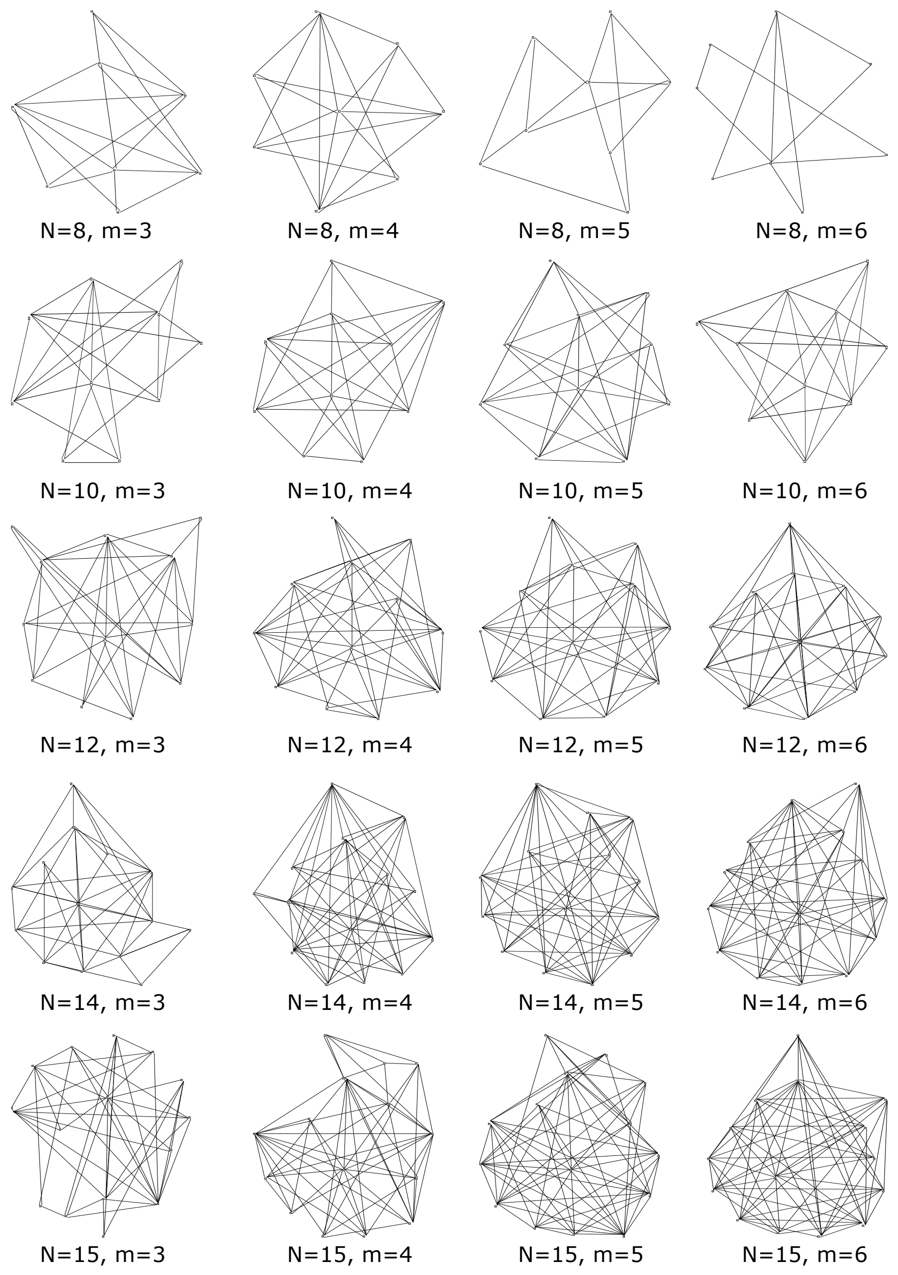


Figure 2.2: Random networks generated using BRITE

The allowable number of iterations to reduce the worst-case network congestion ratio is controlled by I_c. It is decided by considering the allowable computation time, network size, and quality of the solution desired. A larger I_c increases

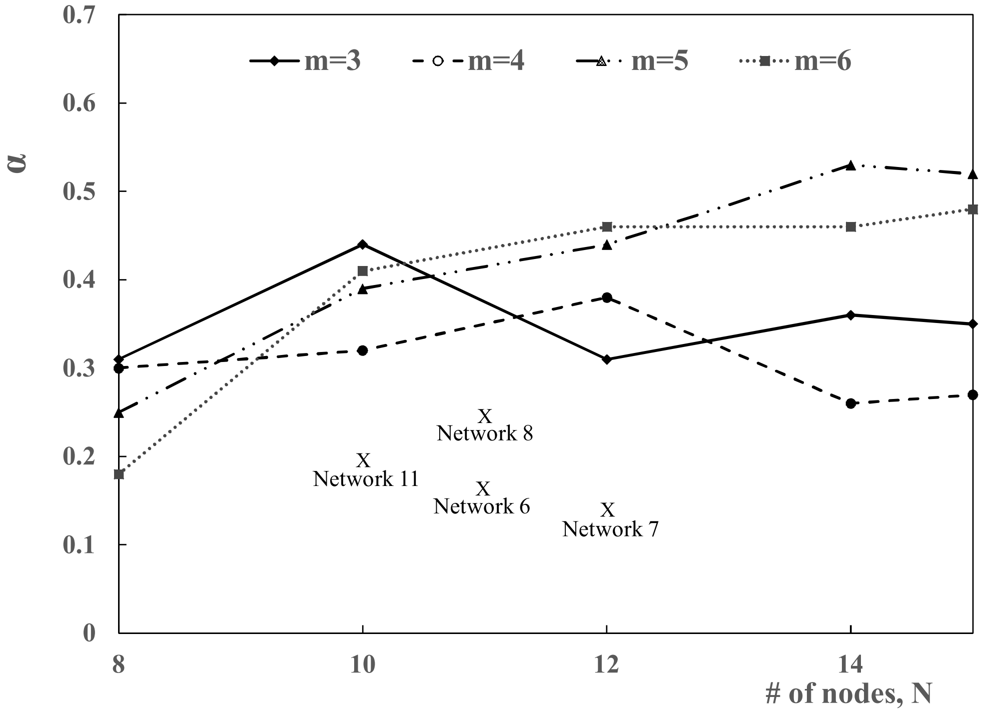


Figure 2.3: α's dependency on number of nodes and adjacency nodes

the chance of getting a solution closer to the optimal one while the computation time increases. Our thorough examination on the effect of I_c suggests that the performance of the presented scheme remains unchanged after I_c reaching an ample high value. Table 2.7 shows the effect of I_c for network 5, as shown in Figure 2.1. The results indicate that the performance of the presented scheme does not change after 1000 even if I_c is increased. It suggests that, 1000 is large enough to converge the solution. We also confirmed the same tendency for other networks in Figure 2.1.

The results in Section 2.5 considers equal-cost multi-path (ECMP) [66] routing is employed. In non-ECMP routing networks, traffic is always routed over a single path, which often results in substantial network resources. On the other hand, ECMP routing distributes the traffic among several *equal cost* paths instead of routing all the traffic along a single *best* path. Fundamentally, ECMP routing can be more efficient compared to single path routing protocols because it can reduce

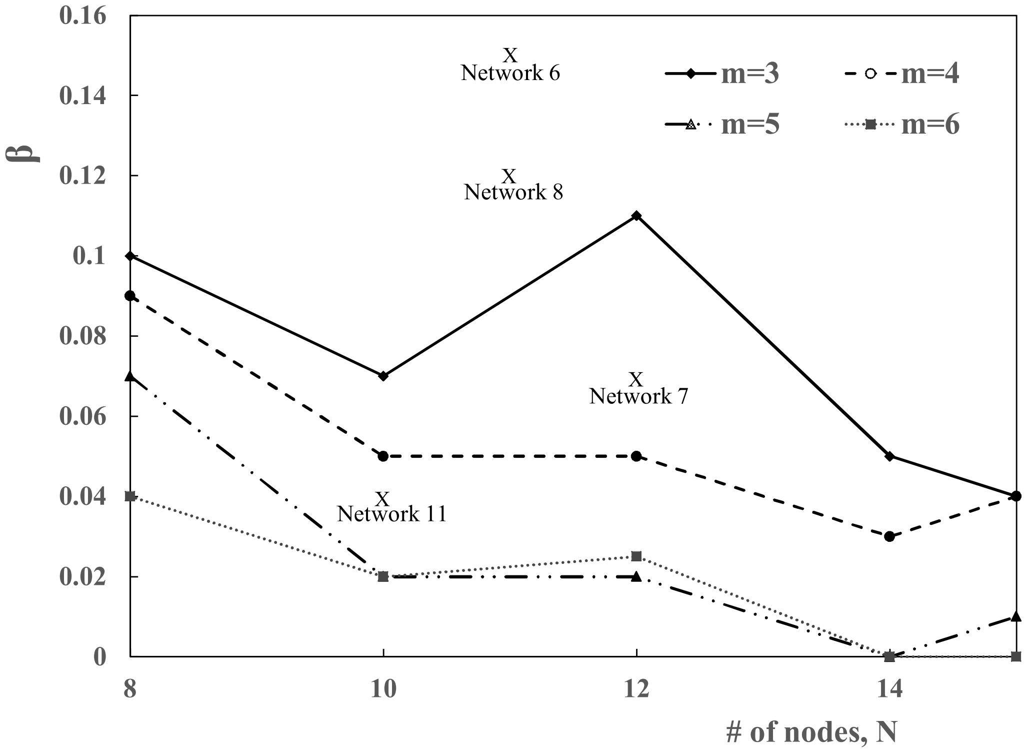


Figure 2.4: β's dependency on number of nodes and adjacency nodes

congestion in "hot-spots", by deviating traffic to unused network resources, thus providing load balancing.

In order to confirm the effectiveness of ECMP routing, the worst-case congestion ratios of ECMP routing are compared with that of single path routing for sample networks 5-12 shown in Figure 2.1. Table 2.8 summarizes the worst-case congestion ratios of ECMP routing and single path routing. The worst-case congestion ratios of ECMP routing is expressed as $r_{PSO-ECMP}$ while $r_{PSO-single}$ expresses the worst-case congestion ratios of single path routing. The worst-case congestion ratios in Table 2.8 are normalized by $r_{PSO-ECMP}$.

ECMP routing's ability to distribute the traffic among several *equal cost* paths is confirmed from Table 2.8. It is observed that the difference between ECMP routing and single path routing increases with average node degree. Higher average node degree increases the number of paths between nodes, thus raising the possibility of multiple equal cost paths in the network. As a result, ECMP rout-

Table 2.6: Comparison of worst-case congestion ratios with different link weight setting schemes

Network	Presented scheme	IPC scheme	min-hop scheme
5	1.00	1.82	2.05
6	1.00	2.45	2.76
7	1.00	3.16	3.74
8	1.00	2.35	2.69
9	1.00	1.83	2.21
10	1.00	3.42	3.52
11	1.00	2.70	2.72
12	1.00	1.58	2.19

Table 2.7: Comparison of presented scheme's performance related to I_c

Value of I_c	Worst-case congestion
100	0.553
200	0.411
300	0.403
400	0.403
500	0.359
600	0.260
700	0.224
800	0.183
900	0.183
1000	0.183
1200	0.183
1500	0.183

ing is able to distribute traffic among these multiple equal cost paths and reduce the congestion.

Table 2.8: Comparison of worst-case congestion ratios of ECMP routing and single path routing

Network	Average node degree	$r_{PSO-ECMP}$	$r_{PSO-single}$
5	3.60	1.00	1.02
6	2.54	1.00	1.02
7	3.00	1.00	1.03
8	4.73	1.00	1.08
9	3.00	1.00	1.02
10	2.78	1.00	1.02
11	5.60	1.00	1.08
12	3.70	1.00	1.04

Chapter 3

Path selection scheme for energy consumption reduction with link failure and traffic uncertainty

3.1 Energy saving routing

How to reduce the energy consumption has recently become a major concern for industries due to economic and environmental reasons. The core network equipments currently occupy 20% of the total energy consumption and it is not showing any sign of slowing down [63]. Therefore, saving energy has also become one of the objectives of network service providers.

In present high speed networks, the link capacities of backbone networks are often over-provisioned in order to permit re-routing when links fail. If the backbone network is already optimized considering link failures, it is possible to reduce the network energy consumption by switching off unused resources or putting them into sleep mode. In today's backbone networks, pairs of nodes are typically connected by multiple physical cables to accommodate more traffic or for future extension purposes. These links are usually over-provisioned only considering network reliability reducing network energy efficiency. Previous studies [64, 65] show that less than 40% of network link capacity is utilized on average, suggesting 60% of the energy consumed by links are wasted on average. Accordingly, shutting down individual unused bundled cables can achieve energy saving in backbone networks. Thus, achieving energy saving by only switching on the minimum number of bundled cables needed for routing in each link even a link failure occurs [38].

It has also become crucial to select efficient link protection mechanisms to limit the impact of failures beside from reducing the power consumption of networks. Present transmission rate of backbone networks exceed 10 Gbits/s and any link failure can adversely affect many systems that rely on the network. As we explained in previous chapters, traffic demand fluctuation is another major problem network service operators worry about.

A path selection scheme that reduces the network energy consumption in the face of link failure and traffic demand fluctuation is needed [38]. In this chapter, we consider how to extend the link weight optimization scheme we presented in Chapter 2 to reduce network energy consumption.

The presented scheme in Chapter 2 reduces the worst-case congestion ratio against any single link failure by finding the most utilized link in the network under single link failure and try to reduce the traffic on that link by re-routing traffic. The network congestion ratio does not express the total resource usage of the whole network, which is more connected to the network energy consumption. We assume that the network energy consumption is expressed as a value that is proportional to the sum of bandwidth used over all the links. Since our goal is to minimize the network energy consumption by minimizing the number of bundled cables used for routing, we set our objective to minimize the sum of bandwidth used over all the links. The objective of the scheme presented in Chapter 2 is to minimize the worst-case congestion ratio which is relatively easy because it is easier to find the most congested link in the network. By increasing the link weight of the most congested link in the network, we can change the worst-case congestion ratio. However, when network energy consumption is considered, we cannot rely on reducing only one link's utilization. We need to change the link weight of the most suitable link which will minimize the sum of bandwidth used over all the links.

3.2 MILP formulation for energy consumption reduction

To reduce the network energy consumption under the hose model traffic, unused cables are deactivated while keeping enough cables to guarantee the routing of all traffic demands. By solving the optimization problem of minimizing the total number of active cables in the network, we can minimize the network energy consumption. The same network model that is considered in section 2.2 with the following additional notations is used in this section. n_{ij} is the number of routers used for routing data from node i to node j. Each link consists of B cables that can be shut down independently. We consider that the traffic is routed among several equal cost paths, also known as Equal Cost Multi-Path (ECMP) routing.

The target of this section is to find the most appropriate set of link weights, W_{min}, for network G that minimizes the worst-case network resource usage over link failure index $l \in F$ and traffic matrices $T \in \tilde{D}$. W_{min} is defined by,

$$W_{min} = \arg \min_{W \in w} \max_{G_l \in \tilde{G}} \max_{T \in \tilde{D}} R(G_l, T, W), \tag{3.1}$$

where $R(G, T, W)$ is the total resource usage for given G, T, l.

Since we are considering the hose model, the problem formulation for generating the worst-case traffic matrices, Equation 2.5, and its dual problem, Equation 2.6, are also true here. Using the dual formulation we can derive the mixed-integer programming formulation of network optimization for minimizing the number of active cables, n_{ij}, in case of link failure under the hose model traffic as shown in Equation 3.2.

$$\min \quad \sum_{(i,j) \in E_l} n_{ij} \tag{3.2a}$$

$$\text{s.t.} \quad \sum_{j:(i,j) \in E_l} x_{ij}^{pq}(W, l) - \sum_{j:(j,i) \in E_l} x_{ji}^{pq}(w, l) = 1,$$
$$\forall p, q \in V, i = p, l \in F \tag{3.2b}$$

$$\sum_{j:(i,j) \in E_l} x_{ij}^{pq}(W, l) - \sum_{j:(j,i) \in E_l} x_{ji}^{pq}(w, l) = 0,$$
$$\forall p, q \in V, i (\neq p, q), l \in F \tag{3.2c}$$

$$\sum_{p\in V} a_p \pi^{ij}(p) + \sum_{p\in V} b_p \lambda^{ij}(p) \leq n_{ij}\frac{c^l_{ij}}{B}, \quad \forall (i,j) \in E_l, l \in F \tag{3.2d}$$

$$x^{pq}_{ij}(W,l) \leq \pi^{ij}(p) + \lambda^{ij}(q), \quad \forall p,q \in V, (i,j) \in E_l, l \in F \tag{3.2e}$$

$$0 \leq f^i_{pq}(l) - x^{pq}_{ij}(W,l) \leq 1 - \delta^{ij}_q(l), \quad \forall p,q \in V, (i,j) \in E_l, l \in F \tag{3.2f}$$

$$x^{pq}_{ij}(W,l) \leq \delta^{ij}_q(l), \quad \forall p,q \in V, (i,j) \in E_l, l \in F \tag{3.2g}$$

$$0 \leq \psi^{jq}(l) + w_{ij} - \psi^{iq}(l) \leq (1-\delta^{ij}_q(l))U, \quad \forall q \in V, (i,j) \in E_l, l \in F \tag{3.2h}$$

$$1 - \delta^{ij}_q(l) \leq \psi^{jq}(l) + w_{ij} \leq \psi^{iq}(l), \quad \forall q \in V, (i,j) \in E_l, l \in F \tag{3.2i}$$

$$n_{ij} \leq B, \quad \forall (i,j) \in E \tag{3.2j}$$

$$f^i_{pq}(l) \geq 0, \quad \forall p,q,i \in V, l \in F \tag{3.2k}$$

$$\delta^{ij}_q(l) \in \{0,1\}, \quad \forall q \in V, (i,j) \in E_l, l \in F \tag{3.2l}$$

$$1 \leq w_{ij} \leq w_{max}, \quad \forall (i,j) \in E_l, l \in F \tag{3.2m}$$

$$n_{ij} = 0,1,2,\cdots, \quad \forall (i,j) \in E_l, l \in F \tag{3.2n}$$

The objective of the above mixed-integer programming formulation is to find the optimal set of active cables in order to minimize the total network energy consumption. The meanings of the constraints are as same as we explained in section 2.3.

Section 3.3 presents the heuristic scheme to reduce energy consumption by deactivating unused cables. Since our goal is to minimize the network energy consumption by minimizing the number of bundled cables used for routing, we set our objective to minimize the sum of bandwidth used over all the links.

3.3 Heuristic approach

The heuristic to reduce the total network energy consumption by deactivating unused cables is similar to the heuristic that is explained in section 2.4. However, the objective function of the heuristic must be changed to a one that considers energy consumption of the network. The sum of individual cost function $\phi(u_{ij}(T))$ of u_{ij} and T over $(i,j) \in E$ and $T \in \tilde{D}$ is chosen as the objective function $F(\tilde{D})$ to reduce the worst-case energy consumption. $F(\tilde{D})$ is defined as,

$$F(\tilde{D}, G_l)_{|\text{for given weights}} = \max_{G_l \in \tilde{G}} \sum_{T \in \tilde{D}} \sum_{(i,j) \in E} \{\phi(u_{ij}(T)) c_{ij}\}, \tag{3.3}$$

where $\phi(u_{ij})$ increases with u_{ij}. For the reasons mentioned in [39], we adopted the following convex piecewise linear cost function for $\phi(u_{ij})$.

$$\phi(u_{ij}) = \begin{cases} \frac{u_{ij}}{c_{ij}}, & 0 \leq \frac{u_{ij}}{c_{ij}} < \frac{1}{3} \\ 3\frac{u_{ij}}{c_{ij}} - \frac{2}{3}, & \frac{1}{3} \leq \frac{u_{ij}}{c_{ij}} < \frac{2}{3} \\ 10\frac{u_{ij}}{c_{ij}} - \frac{16}{3}, & \frac{2}{3} \leq \frac{u_{ij}}{c_{ij}} < \frac{9}{10} \\ 70\frac{u_{ij}}{c_{ij}} - \frac{178}{3}, & \frac{9}{10} \leq \frac{u_{ij}}{c_{ij}} < 1 \\ 500\frac{u_{ij}}{c_{ij}} - \frac{1468}{3}, & 1 \leq \frac{u_{ij}}{c_{ij}} < \frac{11}{10} \\ 5000\frac{u_{ij}}{c_{ij}} - \frac{16318}{3}, & \frac{11}{10} \leq \frac{u_{ij}}{c_{ij}} < \infty. \end{cases} \tag{3.4}$$

3.4 Simulation results

The effectiveness of reducing the worst-case energy consumption of the presented heuristic is evaluated through simulations. The performance of the presented heuristic is compared with that of SO, minimum-hop (min-hop) routing, and IPC. The min-hop routing is a simple, conventional routing scheme which is achieved by setting all the link weights to one. In IPC, the link weights are set so that they are inversely proportional to its capacity. The total network resource usage, R, for routing the data is the performance measure of the evaluation. The sample networks 5-12 in Figure 2.1 are used to determine the characteristics of the presented heuristic. The same simulation conditions that is explained in section 2.5 are used for this simulation, too.

Let $R(l)$ be denoted as the total resource usage for link failure index $l \in F$. The total resource usages of SO, PSO, min-hop, and IPC are denoted as $R_{SO}(l)$, $R_{PSO}(l)$, $R_{min-hop}(l)$, and $R_{IPC}(l)$, respectively.

The worst-case total resource usages, $\max_{l \in F} R_{PSO}(l)$, $\max_{l \in F} R_{SO}(l)$, and $\max_{l \in F} R_{min-hop}(l)$ of the sample networks 5-12 presented in Figure 2.1 for all the single link failure scenarios are as shown in Table 3.1 and they are normalized by $\max_{l \in F} R_{SO}(l)$. For the worst-case network resource usage for single link failure, the following relationship is observed,

$$\max_{l \in F} R_{PSO}(l) < \max_{l \in F} R_{SO}(l) < \max_{l \in F} R_{min-hop}(l). \tag{3.5}$$

This indicates that the proposed scheme can reduce the worst-case network resource usage as compared with SO and min-hop. The achieved reduction rate of the worst-case resource usage, α, is defined as,

$$\alpha = \frac{\max_{l \in F} R_{SO}(l) - \max_{l \in F} R_{PSO}(l)}{\max_{l \in F} R_{SO}(l)}. \tag{3.6}$$

α is also shown in Table 3.1.

Table 3.1: Comparison of worst-case network resource usage for any single link failure scenario

Network	$\max_{l \in F} R_{SO}(l)$	$\max_{l \in F} R_{PSO}(l)$	$\max_{l \in F} R_{min-hop}(l)$	$\max_{l \in F} R_{IPC}(l)$	α
5	1.00	0.52	1.03	1.02	0.48
6	1.00	0.76	1.12	1.29	0.24
7	1.00	0.68	1.05	1.11	0.32
8	1.00	0.82	1.21	1.05	0.18
9	1.00	0.77	1.52	1.63	0.23
10	1.00	0.82	1.78	2.33	0.18
11	1.00	0.64	1.31	1.85	0.36
12	1.00	0.84	1.24	1.42	0.16

The total resource usages, normalized by $R_{SO}(0)$, for no link failure scenario is shown in Table 3.2. For the case of no link failure,

$$R_{SO}(0) \leq R_{PSO}(0) \leq R_{min-hop}(0) \tag{3.7}$$

is observed. When there is no link failure, the total resource usage of the link weight set obtained using PSO may be higher than that of SO. This is because the objective of PSO is to reduce the worst-case network resource usage when a link failure occurs. β is the deviation between $R_{PSO}(0)$ and $R_{SO}(0)$. β is defined as,

$$\beta = \frac{R_{PSO}(0) - R_{SO}(0)}{R_{SO}(0)}. \tag{3.8}$$

β is also shown in Table 3.2. When there is no link failure, β is the *penalty* PSO has to pay to reduce the worst-case network resource usage under any single link failure. The non-normalized simulation results show that the worst-case resource usage of single link failure scenario is significantly larger compared to no link failure scenario.

Table 3.2: Comparison of network resource usage for no link failure scenario

Network	$R_{SO}(0)$	$R_{PSO}(0)$	$R_{min-hop}(0)$	$R_{IPC}(0)$	β
5	1.00	1.21	1.77	1.51	0.21
6	1.00	1.08	1.82	1.89	0.08
7	1.00	1.10	1.90	2.50	0.10
8	1.00	1.16	2.41	1.68	0.16
9	1.00	1.14	1.84	1.91	0.14
10	1.00	1.07	2.15	1.82	0.07
11	1.00	1.11	1.98	1.98	0.11
12	1.00	1.04	1.51	2.04	0.04

In order to understand the relationship between the worst-case resource usage and network topology, the same random network topologies described in Table 2.5 are used. The dependency of α on N and m is shown in Figure 3.1. This result shows that α is increasing with N when m is higher ($m = 5, 6$). It means the difference between the worst-case resource usages of SO and and the presented scheme is increasing. This may relate to the fact that routing flexibility is increased as N and m become higher.

Figure 3.2 shows the dependency of β against N and m. Figure 3.2 indicates that the presented scheme is able to reduce the gap between SO for no link

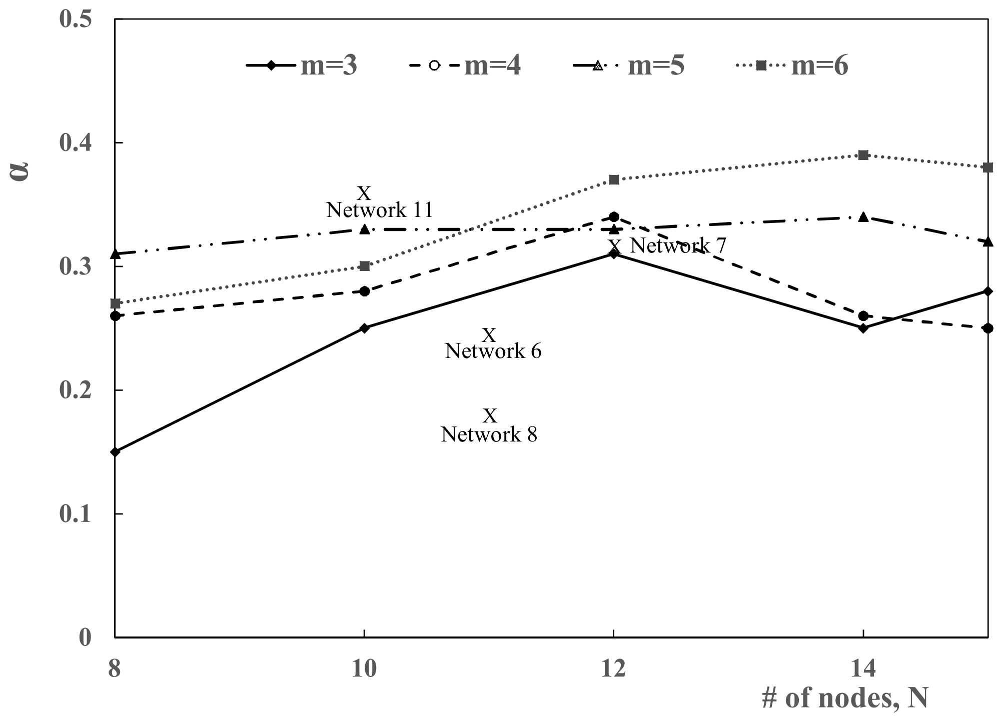


Figure 3.1: α's dependency on number of nodes and adjacency nodes

failure when the network becomes larger. This may occur because there is a wider number of routes to chose from when the network is larger.

As explained in Section 2.5, the value of I_c plays an important role when reducing the worst-case network resource usage. The worst-case network resource usage of network 5 is measured for different I_c values and the results are shown in Table 3.3. The results show that the worst-case network resource usage of the presented scheme is unchanged after 900.

In order to confirm the effectiveness of ECMP routing, the worst-case network resource usages of ECMP routing are compared with that of single path routing for the sample networks 5-12 shown in Figure 2.1. Table 3.4 summarizes the worst-case resource usages of ECMP routing and single path routing. The worst-case resource usages of ECMP routing is expressed as $R_{PSO-ECMP}$ while $R_{PSO-single}$ expresses the worst-case resource usages of single path routing. The results in Table 2.8 are normalized by $R_{PSO-ECMP}$. Table 2.8 shows that the

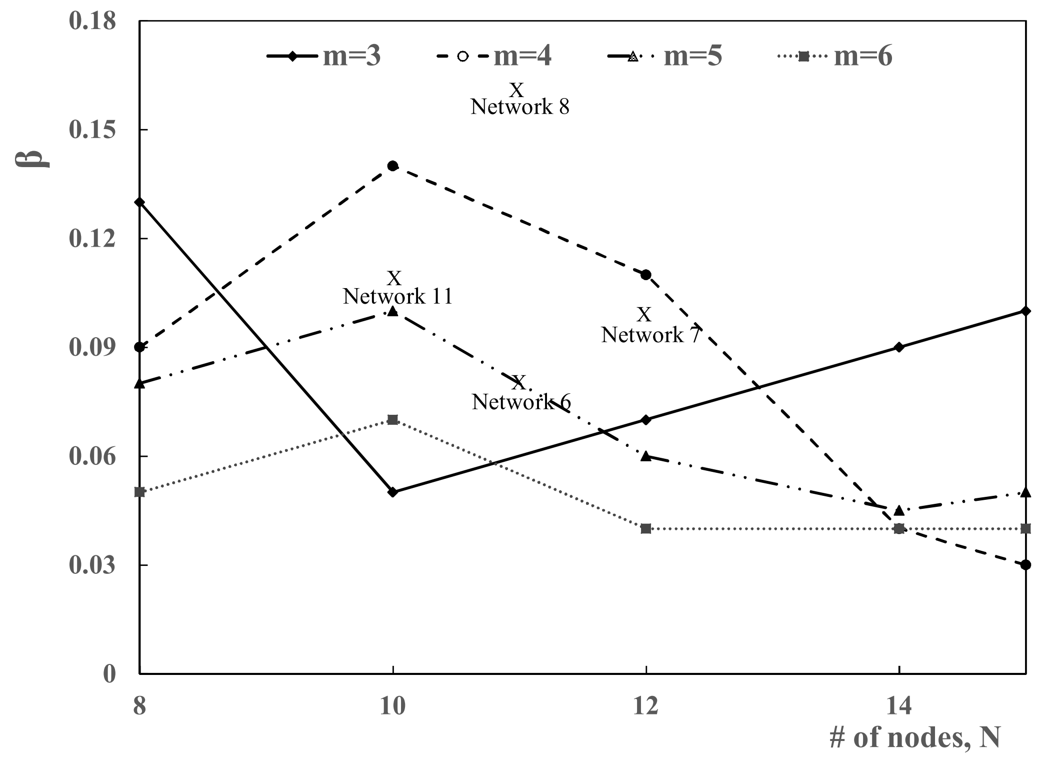


Figure 3.2: β's dependency on number of nodes and adjacency nodes

ECMP routing uses less network resources compared to single path routing in the worst-case scenario.

3.5 Summary

The first part of this thesis presents mathematical formulations to calculate optimal link weight sets that reduce worst-case congestion ratio and worst-case network resource usage by considering the uncertainty of single link failure and traffic demand. The hose model, which relaxes the constraint of network operators to know the exact traffic matrix, is used in this study. This thesis also, presented a heuristic approach to solve the MILP formulations. The worst-case performance of the link weight set calculated using the presented heuristic is equivalent to that of MILP formulation. Numerical results via simulation show that the presented schemes find suitable link weight sets to reduce the worst-case congestion ratio

Table 3.3: Comparison of presented scheme's performance related to I_c

Value of I_c	Worst-case congestion
100	0.325
200	0.319
300	0.294
400	0.201
500	0.181
600	0.174
700	0.152
800	0.129
900	0.108
1000	0.108
1200	0.108
1500	0.108

Table 3.4: Comparison of worst-case resource usages of ECMP routing and single path routing

Network	$R_{PSO-ECMP}$	$R_{PSO-single}$
5	1.00	1.14
6	1.00	1.07
7	1.00	1.12
8	1.00	1.06
9	1.00	1.08
10	1.00	1.12
11	1.00	1.16
12	1.00	1.08

and network resources used in all cases. The presented schemes has to pay a penalty of β for the no link failure scenario. However, if the network administrators want to reduce the worst-case congestion ratio or worst-case network resource usage for any single link failure, presented schemes are better choices.

It is observed that the presented schemes outperform SO for larger networks.

Chapter 4

Path selection scheme for networks with imbalanced node load

The load of nodes is an important factor that affects the overall performance of networks. An overloaded node may decrease the network throughput due to packet queuing. This chapter presents a path selection scheme that is robust against node load. In section 4.1, an overview of Hadoop and its key concepts are introduced. The path selection scheme under node load uncertainty is presented in section 4.2.

4.1 Overview of Hadoop

Hadoop is an open-source implementation of Google's Mapreduce framework and Google File System (GFS). It provides a framework to do parallel-distributed computing with ease thus has become the de-facto platform for processing large-scale data sets, also known as Big Data [44, 67]. A Hadoop cluster generally consists with hundreds to thousands of servers. When considering such a large number of servers, even with today's low level of failure, hardware failures occur on a daily basis [44]. Therefore, Hadoop is designed with mechanisms to provide high level of fault-tolerance while hiding the complexity of distributed computing, scheduling, and communication. All these factors made Hadoop very popular both in industry and academia. Many companies that hold large amounts of data exploit Hadoop to retrieve valuable information from those data. Having access to large amounts of data has become the next big thing in the 21st century making such companies so powerful just because of the future value of the data they hold. Companies like Facebook, Amazon, Yahoo! including Google are early adopters of Hadoop that use Hadoop to process data. Hadoop has grown to be an ecosystem composed of several applications ranging from data warehousing to data flow oriented programming language. Hadoop is used to store, manipulate and extract information from large-scale data sets in many ways.

Although Hadoop has become the de-facto product for large-scale data processing and grown to become an eco-system full of new products being released frequently, it still has room for improvement [68]. Many research efforts are focused on improving data placement, job scheduling, reducing network communication, stage pipelining, in-memory store of intermediate data etc., leaving data read improvement considering server load behind.

4.1.1 Components of Hadoop

Hadoop provides components to store, manipulate and extract information from large-scale data sets efficiently. The Storage layer of Hadoop is called, Hadoop Distributed File System (HDFS) and data manipulation and extraction layer is called, Yet Another Resource Negotiator (YARN). Distributed processing frameworks such as MapReduce, Spark [75], Tez [76] run jobs on top of YARN and HDFS. Typically, both layers are co-located in the same servers. HDFS and YARN both are designed with master-slave architecture to ensure scalability while providing high throughput.

HDFS is responsible for storing data in Hadoop and providing those data to applications when necessary. It is designed for storing very large files with streaming data access patterns running on clusters of commodity hardware [44]. A file in HDFS is broken into blocks and stored dispersedly among servers. Since Hadoop is designed to process large volumes of data, the size of a block is typically large,128MB by default, to reduce disk seeks during data read. A block is replicated to different multiple servers (by default three) for fault-tolerance and availability purposes. The consistency of replicas is not considered because files in HDFS are assumed to be append-only and the blocks are written once and read many times. Write-once-read-many means that once data is written to HDFS, that particular data will be read by various processing tasks many time over the time. Therefore, improving how data is read in HDFS has a larger impact on the overall performance of HDFS compared to improving how data is written.

Data locality is the property that defines whether data and processing task are co-located on the same server. Hadoop tries to co-locate data and processing tasks so that data access is fast because data is local [44]. This is one of the revolutionary concepts that was introduced in Hadoop: "taking calculation to where data is" rather than "taking data to the calculation". Unfortunately, it is not always possible to co-locate data and processing tasks due to resource unavailability. Experiments done by Ibrahim et al. [69] show that approximately 23% of the map tasks are non-local map tasks. This means 23% of the map tasks read or fetch non-local data from a remote server that holds a copy of the data before starting to process the data. Many improvements for enhancing data placement (data write) [70, 71, 72] in HDFS were presented. However all of these are focused on optimizing replica placement, which means writing data into HDFS, while paying no attention to data read improvement, that occurs more often compared to data write.

HDFS has two daemons operating in a master-slave architecture. The master components is a NameNode and the slave component is a DataNode. The NameNode oversees and manages data storage. It is responsible for storing the file system tree, and the metadata of all the files and directories in HDFS. It also maintains information about the locations of the blocks of a specific file in the memory for faster response of block locations. DataNodes store and retrieve data blocks when requested by clients or the NameNode, and they periodically report the blocks list they store to the NameNode.

YARN is responsible for managing cluster resources and scheduling jobs in the cluster. It provides APIs for requesting and working with cluster resources for distributed processing systems. Figure 4.1 shows the relationship of distributed processing systems, YARN, and HDFS.

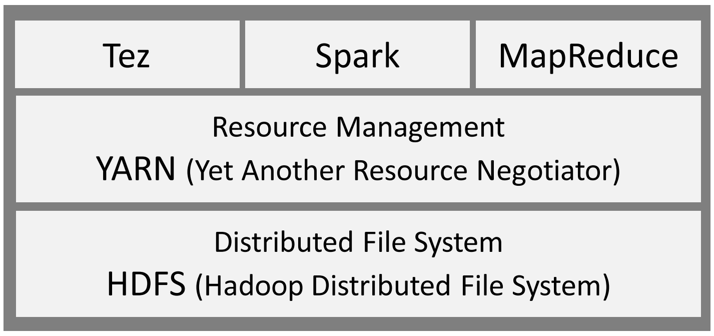

Tez
Spark
MapReduce
Resource Management
YARN (Yet Another Resource Negotiator)
Distributed File System
HDFS (Hadoop Distributed File System)

Figure 4.1: Components of Hadoop and their relationship

YARN also consists with two daemons, a ResourceManager and a NodeManager. ResourceManager is the master component and it manages the resources within the cluster and schedule jobs. The slave component of YARN, NodeManager, launches and monitor tasks which are run within containers. The master daemon of HDFS, NameNode, and the master daemon of YARN, ResourceManager, are usually co-located in the same servers called master servers. The slave daemons of HDFS, DataNode, and the slave daemons of YARN, NodeManager, are also co-located in slave servers.

4.1.2 MapReduce framework

The MapReduce framework [42] is a distributed processing system that utilizes a set of servers to process large volume of data. It employs the data flow programming model that data flow through map phase, shuffle phase, and reduce phase. Map task executes a map function defined in a program and read a data block as input data which contains data in key-value pairs. The map task combines the key-value pair records with the same key as a tuple and write the output to an intermediate file in the local file system. The shuffle phase sort the tuples in the intermediate files and transfer the data to a reduce task. After receiving all intermediate data by the reduce task, it applies the reduce function defined in the program to all the intermediate data. The result of the reduce task is written to an output file in HDFS.

4.1.3 Rack awareness

Typically, large clusters with hundreds of servers are organized in hierarchical multi-path networks [77]. Fat-tree topology [78] is a widely used topology for such networks. Figure 4.2 shows a typical Hadoop cluster, that is configured in fat-tree topology with rows of racks. Each rack contains 20-40 servers and the servers are connected to a top-of-rack switch. Furthermore, multiple paths and different host counts between two servers can exist making path selection a critical factor in communication between servers. The top-of-rack switches connect to one or more core switches, creating multiple paths between two servers. Traffic that go from one rack to another, also called cross-rack traffic, travels through the core layer links, making them a bottleneck. The links which connect core layer and edge layer become a bottleneck because they are shared by more servers at the same time.

Hadoop is designed to reduce cross-rack traffic and utilize in-rack resources as much as possible. Hadoop uses a concept called called "rack awareness" to achieve this. Unfortunately, Hadoop is not able to understand the network topology by itself without any human help. Cluster administrators have to assign rack locations of each server, especially slave servers, so that Hadoop is able to utilize in-rack resources to the max. When rack information is not configured, Hadoop considers the cluster topology as flat and all the servers are in a single rack. Hadoop calculates the static network distance or the closeness between slave servers using this rack information. Two servers in the same rack are closer compared to two servers in separate racks. The calculated network distance using rack information will be static as long as the rack layout and rack information are not changed. Additionally, the calculated network distance does not consider server load since it is calculated by using static rack configuration information only.

4.1.4 Data read procedure in HDFS

Rack configuration information is utilized in many parts of Hadoop, task scheduling, replica placement, data read etc. When a client reads data from HDFS, it has the choice of selecting one DataNode among the three DataNodes that hold

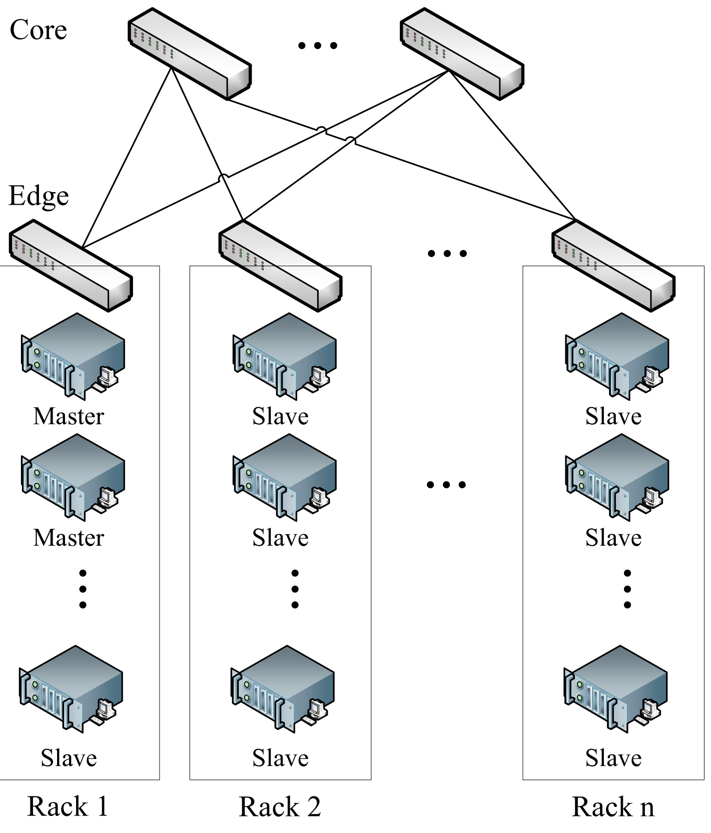


Figure 4.2: Fat tree topology based Hadoop cluster

the replicas. This selection of DataNode is done by the NameNode using rack configuration information, which is transparent to the client. The data reading procedure of HDFS is explained below.

1. The client that wants to read data contacts the NameNode to determine the block locations.

2. The NameNode finds the addresses of the DataNodes that hold requested data using metadata and block reports from DataNodes. Then the NameNode sorts the addresses of DataNodes according to the proximity to the client using rack information and sends the sorted list of addresses to the client which requested block locations.

3. If the client is itself a DataNode and holds a replica of the data, it reads data from the local disk directly. Otherwise, the client connects to the closest DataNode (first DataNode in the list) according to the sorted addresses it received from NameNode and fetches data over the network.

Hadoop only considers the rack configuration information when selecting a DataNode. Rack configuration information does not reflect the server load. This static behavior can lead to longer data read time compared to a dynamic behavior [79]. Consider the case where the closest DataNode decided by the NameNode is overloaded with processing other tasks and other two DataNodes that hold the same data block is relatively free. Then, the data fetch request by the client to the overloaded DataNode will be queued making the data read time longer. Therefore, DataNode selection scheme which can take account of the server load is desirable. Selecting one DataNode among three DataNodes can be interpreted as selecting a path from three paths as we explained in section 1.5.

4.2 Path selection scheme under node load uncertainty

This section presents the path selection scheme under server load uncertainty for Hadoop clusters. We use the term "server" to represent a "node". The presented scheme is based on delay distribution between servers and we assume that the delay time of a server represents the server load (queuing delay). In order to calculate the delay distribution between servers, we periodically measure the round-trip time (RTT) between DataNodes. The delay distributions between DataNode pairs are compared when selecting a DataNode to fetch data from. The presented scheme adds two changes to Hadoop.

1. Adds a new feature to DataNode process to measure round-trip time between DataNodes.

2. Extends HDFS data reading procedure to use RTT-based delay distribution.

Conventional Hadoop calculates a network distance between servers by using the physical rack layout information only. Even if the "static" physical distance such as hop count is used, we still have a problem. Since server load fluctuates frequently, it is necessary to dynamically calculate the best server according to the state of the server's load. That is why we use delay distribution which can be calculated by using commonly available/obtainable information in any environment. Instead of using a static metric such as hop count, the presented scheme uses RTT-based delay distribution to calculate the logical distance between servers [73, 74]. Authors in [80, 81, 82, 83, 84] have used RTT for server selection by considering the server load. In [84], it is stated that RTT is perhaps the best metric to measure server load and network delay. We adopt RTT not only for these reasons but also it is relatively easy and inexpensive to use in any environment. Using RTT to measure the delay distribution requires its periodic measurement. RTT measurement can be done using an *active* approach (i.e. sending probe packets) or with a *passive* approach (i.e. trying to calculate RTT based on existing traffic). In theory, the passive approach is preferable since it does not introduce any additional traffic into the network. On the other hand, measuring RTT based on existing traffic can be more complex and CPU intensive [85]. Additionally, results in [86] indicate that the overhead added by periodic RTT measurement was less than 1% in terms of additional network traffic. Therefore, we decided to focus on the *active* approach.

The procedure of measuring RTT between servers is explained below.

1. At startup, each DataNode gets a list of DataNodes connected to the cluster.
2. Each DataNode sends an echo request of Internet Control Message Protocol (ICMP) to all the other DataNodes periodically.
3. All the DataNodes record the time it takes to get an ICMP echo reply (RTT) from other DataNodes since sending the echo request.

RTT can suddenly change depending on the network traffic and server workload. In order to minimize the impact of the sudden changes of RTT, the distribution of RTT, or delay distribution that shows the probability characteristic of the delay is used [73, 74]. Delay distribution between DataNodes can be calculated by using the periodically measured RTT. Multiple measurements of RTT is required to represent the actual delay characteristic between servers. Therefore, the RTT is measured right from the start of the cluster and the delay distribution is updated periodically, whenever the RTT is measured. We update the delay distribution by using exponential smoothing technique [87, 88, 89] as shown in Eq. (4.1). Exponential smoothing technique is often used for time-series data and it can be easily applied for making some determination based on prior observations. Let t be the measured RTT between two servers and $p_\tau(t)$ be the measured RTT distribution at time τ. $f_\tau(t)$ is the smoothed RTT distribution at time τ, $f_{\tau-1}(t)$ is the smoothed RTT distribution calculated at time $\tau-1$, and α is the smoothing factor.

$$f_0(t) = p_0(t), \tau = 0 \tag{4.1a}$$

$$f_\tau(t) = \alpha p_\tau(t) + (1-\alpha) f_{\tau-1}(t), \tau > 0 \tag{4.1b}$$

$$0 \leq \alpha \leq 1 \tag{4.1c}$$

In other words, the smoothed RTT distribution $f_\tau(t)$ is a simple weighted average of the current measurement of RTT distribution and the previous smoothed RTT distribution. It is required to have multiple RTT measurements to understand the actual delay distribution between server pairs. The value selected for α determines how $f_\tau(t)$ is updated. Larger values of α have less of a smoothing effect and give greater weight to recent changes in the measured data. In the limiting case with $\alpha = 1$ the output series is same as the RTT distribution. Smaller values of α have more of a smoothing effect and give greater weight to measurements from the more distant past.

Delay distribution can be used as a metric to compare server load. However, it is necessary to compare similar points of the delay distributions. As the comparison point of delay distribution, we can select the average time, maximum time, minimum time, or a randomly selected point of the delay distribution. This study uses the minimum, average, maximum, ϵ, and $1-\epsilon$ (for a given ϵ) as comparison policies. t_{min} is the minimum delay time, t_{max} is the maximum delay time, and

t_{avg} is the average delay time. ϵ is the percentile of delay distribution and it is defined as,

$$\epsilon = \int_{t_\epsilon}^{t_{max}} f_T(t)dt. \tag{4.2}$$

By using Equation (4.2) for a given ϵ, we can calculate the t_ϵ. Similarly, $t_{1-\epsilon}$ is defined as Eq. (4.3) and $t_{1-\epsilon}$ can be calculated for a given ϵ.

$$\epsilon = \int_{t_{min}}^{t_{1-\epsilon}} f_T(t)dt \tag{4.3}$$

The relationship between t_{min}, t_{avg}, t_{max}, t_ϵ, and $t_{1-\epsilon}$ are shown in Figure 4.3.

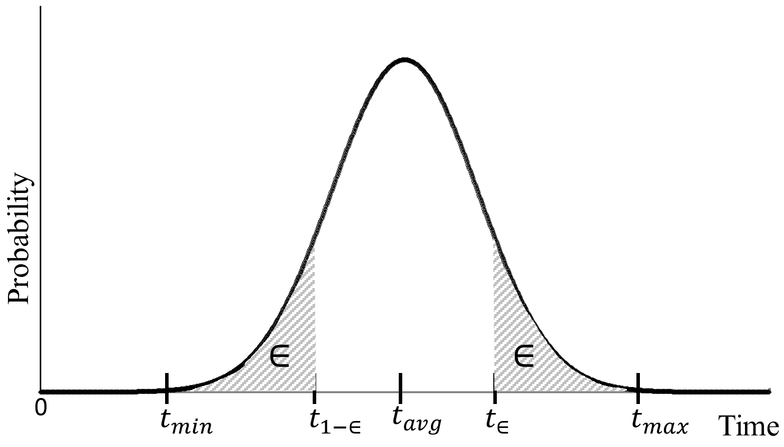


Figure 4.3: Relationship between delay distribution comparison policies

The modified data reading procedure of HDFS utilizing the delay distribution is explained below.

1. A client contacts the NameNode to determine the locations of the data.
2. The NameNode sends the addresses of the DataNodes that have a replica of requested data to the client.
3. If the client is itself a DataNode and holds a replica of the data, it reads data from the local disk directly. Otherwise, the client sorts DataNodes list according to the delay distribution comparison policy and connects to the DataNode with the least delay to fetch non-local data.

The selected remote DataNode for a particular comparison policy can be expressed as,

$$DataNode(i) = \arg\min_{i' \in M} t_(i'). \qquad (4.4)$$

Chapter 5

Experimental results of path selection scheme for networks with imbalanced node load

This chapter presents the experimental results of path selection scheme for networks with imbalanced node load. The results in this chapter are not numerical simulation results but experimental results obtained running sample jobs on real Hadoop clusters. Experimental results are closer to real-world situations compared to numerical simulations. Thus, they are presented as a separate chapter. The experiments were done using Hadoop clusters created on a cloud platform.

In section 5.1, the environment used for experiments is explained. Section 5.2 introduces the results followed by the summary in section 5.3

5.1 Experiment environment

The experiments are done using Hadoop clusters created on AWS EC2 instances [91]. Public cloud environments, such as AWS, are designed as highly-multiplexed, shared environments with virtual servers and tasks from numerous tenants co-existing in the same physical server to achieve cost effectiveness and on-demand scaling. This shared nature of resources make the load of servers unpredictable thus a suitable platform to confirm the effectiveness of the presented path selection scheme. In order to implement the presented scheme, a new Java class is added to the DataNode daemon to measure the delay time between DataNodes and *DFSClient* class is improved to compare the delay distribution.

Six clusters, one cluster for each policy described in section 4.2, are used for the experiments and their configurations are described as follows. All the clusters are based on CDH (Cloudera's Distribution including Apache Hadoop) 5.11.1 [92] and deployed on AWS EC2. Each cluster consists of seven compute optimized c4.4xlarge [91] instances (virtual servers) with one master server and six slave servers. The master server contains the NameNode and the ResourceManager daemons while each slave server containing the DataNode and the NodeManager daemons. Each instance has 16 vCPUs, 30 GiB of memory with EBS storage and they are summarized in Table 5.1. CentOS 7.3 is installed as the operating system. CDH 5.11.1 includes Hadoop-2.6 with backports of latest patches. Default scheduler of CDH, *FairScheduler* [93], is used without configuring any additional queues. 2GB of memory and one virtual CPU for each map and reduce task is configured. The maximum memory allocated for YARN containers in each nodemanager is 28GB, leaving 2GB of memory for the operating system. Similarly, 15 virtual CPUs are allocated for YARN containers in each nodemanger leaving one virtual CPU for the operating system tasks. A total of 168GB of memory and 90 virtual CPUs are available in the cluster. Rack assignment is not configured for any of the servers since we do not have any physical rack layout information of the public cloud environment. Hence, Hadoop assumes that all the servers are in one rack named *default*. For performance evaluation, we create random text data using Hadoop's *randomtextwriter* program. All the jobs are executed at the same time from six clusters to limit the impact of load imbalance and network delay depending on the time. Each job is executed ten times and the averaged results are shown section 5.2.

5.2 Results and discussions

This study compares the HDFS data read performance of the presented scheme with conventional Hadoop by fetching data in HDFS and running MapReduce jobs. The total time took to fetch data from HDFS and MapReduce job run time are the performance measures of the evaluation.

Table 5.1: Virtual server specifications

Resource type	Value
vCPU	16 cores
CPU	Intel Xeon E5-2666 v3
Clock speed (GHz)	2.9
Memory (GiB)	30
Disk (GB)	300

In this experiment, the value of α is set to 0.5 in order to consider both new and old probabilities of RTT. RTT between DataNode pairs is measured every second. ϵ is set to 0.15 after comparing data fetch times for different values of ϵ. Measured data fetch times for different values of ϵ are shown in Table 5.2. Data fetch times are average values of ten data fetches and the deviation of the results were under 10%.

Table 5.2: Data fetch times of different ϵ values

Value of ϵ	Data fetch time [sec]
0.00	5.3203
0.05	5.1034
0.10	4.8923
0.13	4.2532
0.15	4.2450
0.17	4.3429
0.20	4.8624
0.25	4.7432
0.30	4.8432

First, the time it takes to fetch data from HDFS using the presented scheme and conventional Hadoop is measured. The averaged results are shown in Table 5.3. A 1GB file (eight data blocks) from HDFS is fetched to one of the DataNodes using the built in *hdfs dfs* command. For this experiment, we select the DataNode that has the least number of replicas of the 1GB file so that Hadoop can fetch more replicas from remote DataNodes. In this experiment, there are only one replica stored at the particular DataNode and seven replicas are fetched from remote DataNodes. Data fetch time for two scenarios are measured: without any background jobs running and with background jobs running.

To simulate a real-world multi-user cluster, data fetch time while running background jobs are measured. Three *wordcount* MapReduce jobs [50, 69, 77], each counting words in separate 20GB files are used as background jobs. $Wordcount$ is selected because it achieves a balance in both map and reduce stages. Each map task of the *wordcount* job reads the input file, line by line and breaks it into words with key/value pair of the word and 1. Each reduce task sums the counts of each word and creates a single key/value with the word and sum as the result of the job.

Conventional Hadoop is expressed as T_{con}. $T_{policy}, policy \in \{avg, max, min, \epsilon, 1-\epsilon\}$ shows which policy is used as the delay distribution comparison policy. T_{avg} expresses that t_{avg} is used as the delay distribution comparison policy, T_{max} expresses that t_{max} is used as the delay distribution comparison policy, etc.

Table 5.3: HDFS data fetch times

Policy	Time [sec]	
	Without background jobs	With background jobs running
T_{con}	4.3988	5.7020
T_{avg}	4.2700	5.0283
T_{max}	4.3587	5.5263
T_{min}	4.3454	5.5682
T_{ϵ}	4.2366	5.1521
$T_{1-\epsilon}$	4.3131	5.3139

When there are no background jobs,

$$T_{\epsilon} < T_{avg} < T_{1-\epsilon} < T_{min} < T_{max} < T_{con} \tag{5.1}$$

is observed. For the scenario with background jobs running,

$$T_{avg} < T_{\epsilon} < T_{1-\epsilon} < T_{max} < T_{min} < T_{con} \tag{5.2}$$

is observed. In the case that there are background jobs running, all the policies including conventional Hadoop take longer time to fetch data compared to the result of no background jobs. When there are background jobs running, three jobs process 60GB of data. It means that the server load is higher compared to the scenario where no background jobs are running and there are more traffic (non-local data fetch of map tasks, shuffle data etc.) transferred between the DataNodes. The higher server load in DataNodes and large traffic in the network from the background jobs causes the data fetches to take longer time compared with no background jobs scenario.

Hadoop's conventional data fetch mechanism is used in T_{con}. Therefore, when fetching non-local data from remote DataNodes, it randomly selects a remote DataNode to fetch data from. This randomly selected DataNode server (or the physical server that hosts the virtual server) might be overloaded or free depending on the workload at that time. Equations (5.1) and (5.2) both show that this randomly selected remote DataNode is not the best DataNode to fetch data from. On the other hand, $T_{policy}, policy \in \{avg, max, min, \epsilon, 1-\epsilon\}$ respectively uses t_{avg}, t_{max}, t_{min}, t_{ϵ}, and $t_{1-\epsilon}$ of delay distribution as a unit to measure the server load and selects the server with least server load, resulting shorter data fetch time.

When there are no background jobs running,only a small number of remote data fetching from the same DataNode will occur. Therefore, reducing the worst-case delay time, t_{max}, will finish the data fetch in the least amount of time. However, t_{max} and t_{min} are the two extremes of delay distribution and they do not accurately represent the delay characteristic of the cluster. Additionally, t_{max} and t_{min} are relatively unstable and sometimes contain abnormal delay times due to sudden server load or network traffic fluctuations in the cluster which is deployed on a public cloud environment. Therefore, a policy that is robust against sudden server load fluctuations while reducing the worst-case delay is desirable. T_{ϵ} is more suitable since it reduces the delay times that are $\epsilon\%$ from the worst-case delay, which is relatively stable compared to t_{max} or t_{min}. The experimental results also show that T_ϵ is able to fetch data in a shorter time compared to conventional Hadoop and other policies when there are no background jobs.

When there are background jobs running, background jobs also fetch data in addition to the data fetch command that we run, leading to multiple data fetches from the same remote DataNode. When there are multiple data fetches from the same remote DataNode, reducing one data fetch does not shorten the total data fetch time. This is because it is difficult to estimate which data fetch should be reduced. A policy that reduces the total time of multiple data fetches at the same time is more suitable. T_{avg} is more suitable since it reduces the average delay time of multiple data fetches. The experimental results also show that T_{avg} is able to fetch data in a shorter time compared to conventional Hadoop and other policies when there are jobs running in the background.

Table 5.4 shows the averaged results of *wordcount* MapReduce job, processing 10GB file without any jobs running in the background. Rack-local map tasks are the map tasks that fetch data from other DataNodes. In this experiment, only data-local and rack-local map tasks exist. The reason for that is, Hadoop considers as all servers in a single rack since rack locations are not configured. The number of rack-local map tasks are verified from the job counters information.

Table 5.4: Job completion time without background jobs running

Policy	Time [sec]	Average ratio of rack-local map tasks [%]
T_{con}	85.294	19.89
T_{avg}	80.551	19.89
T_{max}	83.966	20.00
T_{min}	84.880	20.11
T_ϵ	80.053	20.00
$T_{1-\epsilon}$	82.851	20.11

From Table 5.4,

$$T_\epsilon < T_{avg} < T_{1-\epsilon} < T_{max} < T_{min} < T_{con} \tag{5.3}$$

is observed. Equation (5.3) shows that the presented scheme is able to reduce the *wordcount* job run time compared to conventional Hadoop even though the average ratio of rack-local map tasks is slightly higher. In the case that there are no background jobs running, *wordcount* job is finished in the shortest time when T_ϵ is used. This is similar to what we observed in Equation (5.1). We can say that a policy which reduces the worst-case delay time while being robust to sudden traffic changes is preferable when there are only a few data fetches occurring in the cluster.

Table 5.5 shows the averaged results of *wordcount* job, processing 10GB file while background jobs are running. Three *wordcount* jobs are executed in background, processing separate 50GB files. Table 5.5 also shows the average rack-local map task ratio.

Table 5.5: Wordcount job completion time without background jobs running

Policy	Time [sec]	Average ratio of rack-local map tasks [%]
T_{con}	167.931	30.40
T_{avg}	152.293	30.00
T_{max}	159.902	31.32
T_{min}	159.591	30.04
T_ϵ	153.417	30.00
$T_{1-\epsilon}$	156.081	30.32

From Table 5.5,

$$T_{avg} < T_\epsilon < T_{1-\epsilon} < T_{min} < T_{max} < T_{con} \tag{5.4}$$

is observed. This shows that the presented scheme is able to reduce the job runtime even when there are background jobs running. The average job run time compared to Table 5.4 increases for all the policies including conventional Hadoop. This is related to the fact that there are more tasks running in the cluster causing more load on each DataNode and more traffic in the cluster. Equation (5.4) shows that T_{avg} finishes in the shortest time. This is similar to what we observed in Equation (5.2). Therefore, the policy that reduces the average delay time is most suitable for real-world workloads where there are multiple non-local data fetches occurring at the same time.

In real-world Hadoop clusters, there are multiple jobs running at the same time. From the above experimental results it is confirmed that reducing the average of the delay distribution is most effective in such clusters. In order to further investigate the effectiveness of the policy that compares average delay distributions, data fetch times are measured while changing the number of background jobs. Table 5.6 shows the data fetch times of conventional Hadoop and the most effective policy in the proposed scheme, which reduces the average delay time. A 1GB file is fetched from HDFS to measure the data fetch times. $Wordcount$ MapReduce jobs that process a 20GB file each are used as background jobs.

Table 5.6: HDFS data fetch time with changing the number of background jobs

Number of	Data fetch time [sec]		Data fetch time
background jobs	T_{con}	T_{avg}	reduction rate
0	4.3988	4.2700	0.029
1	4.9221	4.6842	0.048
3	5.7020	5.0283	0.118
5	9.7285	87344	0.102
8	10.7371	9.5494	0.111
10	11.9826	10.8118	0.098

$$T_{avg} < T_{con} \tag{5.5}$$

is observed from Table 5.6. These results further prove that the proposed scheme is effective and continues to outperform conventional Hadoop with the number of background jobs increasing. However, the gap between conventional Hadoop and proposed scheme stops increasing after three background jobs. This is mainly related to the network throughput reduction (network delay increase) due to the increase in number of background jobs. More background jobs cause more traffic in the network, reducing the network throughput. As a result, the time it takes to transfer data over the network (network delay) increases similarly for both conventional Hadoop and the proposed scheme. However, the block locations and the number of blocks of the 1GB file are fixed and the time to fetch non-local data of the 1GB file becomes identical. Therefore, the effectiveness of the proposed scheme peaks irrespective of the number of background jobs.

Furthermore, data fetch time is measured while changing the size of the fetched data. Three *wordcount* jobs are used as background jobs. In real-world clusters there are multiple jobs running at the same time and from the above experimental results it is confirmed that reducing the average of the delay distribution is most effective in such clusters. Therefore, we only utilize the policy that compares the average delay distribution. Table 5.7 summarizes the data size, data fetch time, and data fetch time reduction rate of the proposed scheme and conventional Hadoop. Results from Table 5.7 show that the presented scheme becomes more effective as the fetched data size increases. The number of data blocks that needs to be fetched from remote DataNodes increases with the data size. More data blocks provide more opportunities for the proposed scheme to expand the difference with conventional Hadoop.

Table 5.7: Comparison of data fetch time with fetched data size

Data size [GB]	Data fetch time [sec]		Data fetch time
	T_{con}	T_{avg}	reduction rate
1	5.7020	5.0283	0.118
3	14.8148	12.9987	0.123
5	20.5169	17.9520	0.125
10	57.9539	50.6240	0.127
15	105.3342	93.2251	0.127

5.3 Summary

We evaluated the performance of the presented path selection scheme for networks with uncertain node load. The experimental results show that the presented scheme, which uses delay distribution as an alternative to compare the server load between servers, is able to fetch non-local data from remote DataNodes efficiently compared to conventional Hadoop. This results in shorter job run times, saving cluster resources which can be used for other data processing jobs. Equations (5.2) and (5.4) resemble the results that are closer to real-world multi-user clusters where there are tens or hundreds of jobs running simultaneously. The number of non-local data fetches increases with the number of concurrent jobs since it is harder to schedule processing tasks on the same server where data is. Therefore, T_{avg}, which reduces the average delay time, is more suitable for real-world clusters. Moreover, the results suggest that the data fetch time difference between the above policy and conventional Hadoop widens as the data size increases.

Chapter 6

Conclusions and future works

6.1 Conclusions

In order to provide a high quality service to customers, robust and efficient path selection schemes are desired. This thesis presented robust path selection schemes under traffic demand uncertainty, link failure uncertainty, and node load uncertainty.

In the first part of the thesis, a path selection scheme for Internet Protocol (IP) networks with traffic demand and link failure uncertainty is presented. Open Shortest Path First (OSPF) is widely used as a routing protocol in IP networks and it selects the shortest path between source and destination pairs. Shortest paths are determined by link weights that are configured by network operators. It is possible to calculate robust paths that satisfy an objective set by the network operator by optimizing the link weights. There have been no studies that consider both traffic demand uncertainty and link failure uncertainty in IP networks in literature.

The presented scheme considers the hose model, which requires specifying the total egress and ingress traffic at each node only, and PSO to optimize link weights that can handle traffic demand fluctuation and link failure. Network operators may have different objectives when setting paths. In order to address these objectives, the link weight optimization scheme is applied to reduce the worst-case congestion ratio and the worst-case network resource usage. The network resource reduction is achieved by switching off unnecessary cables thus saving energy. The network congestion reduction and the network resource reduction problems are formulated as mixed integer linear programming (MILP) problems that consider both traffic demand fluctuation and link failure. Due to the difficulty of solving the MILP formulation for larger networks, heuristic approaches are also presented. The heuristics significantly reduces the computation complexity as compared to the MILP formulations while keeping a comparable performance. They efficiently selects the worst-case performance traffic matrix and reduces the objective function as compared to a brute-force scheme that is computationally expensive when searching the link weight space against all the possible traffic matrices and topologies created by single link failures. Simulations based on both real networks topologies and random networks show that the presented heuristic schemes, while being robust to traffic demand fluctuations and link failures, are able to reduce the worst-case network congestion ratio and the worst-case network resource usage. The presented scheme uses random link weights to generate initial traffic matrices in the first stage. Therefore, the worst-case results may slightly change depending on the set of link weights generated randomly in the first stage.

In the second part of the thesis, a path selection scheme considering node load is presented. Overloaded node will queue requests by other nodes reducing the overall system performance. Hadoop, a parallel-distributed framework, depends on the network to run jobs among multiple servers efficiently. However, it does not consider the server load when selecting a server to fetch non-local data. This thesis presented a path selection scheme based on delay distribution between servers for Hadoop clusters to select a DataNode when fetching non-local data. In order to understand each server's workload dynamically, it periodically calculates the delay time between servers. Then it selects one server by comparing the delay distributions between server pairs. The experiments done using real Hadoop clusters deployed on a public cloud environment observe that the presented scheme, while being robust to server load, is able to select the best path resulting shorter data fetch time compared to conventional Hadoop. This reduction in data fetch time will lead to the reduction in job runtime, especially in real-world multi-user clusters where non-local data fetching can happen frequently.

6.2 Future work

The work presented in this thesis opens the ways to several directions for future work.

In this thesis, the probability of a link failure is considered to be equal for all the links in the network and given. However, in reality, the failure probability of each link may vary and network operators can calculate the failure probability of each link using historical data. This historical data can be fed to a Deep Learning neural network to create a model, which learns the characteristics of the link failures. The created neural network model can be used to predict the probabilities of each link failure and this work can be extended to optimize link weights using the calculated probability of link failures, called stochastic optimization [95], considering traffic demand fluctuation.

The first part of this work considered traffic demand and link failure uncertainties while the second part considered node load uncertainty. A path selection scheme that considers both link failure and node load uncertainty is left as future work.

Both parts in this work considered conventional networks. Software Defined Networks (SDN) [96] is a promising topic studied by researches all over the world in recent years. SDN technologies provide more insight into the operational status of the network. This allows network operators to take quick actions depending on the network status. Applying traffic demand, link failure, and node load uncertainties to SDN is a topic we want to investigate in the future.

Malicious attacks can adversely affect the performance of a network. We want to consider traffic demand uncertainty with malicious attacks as another future work.

In the second part of this thesis, a path selection scheme for Hadoop clusters under node load uncertainty is presented. There are three replicas of data in HDFS. However, Hadoop only uses one of those replicas at a time when reading data for processing. Expanding Hadoop to read data from multiple replicas in different ratios at the same time can lead to better data read times. Investigating how to implement this method can be an interesting research topic.

References

[1] D. Mehdi and K. Ramasamy, “Network Routing: Algorithms, Protocols, and Architectures 2nd Edition,” Elsevier, Sep. 2017. 1, 2, 3, 4, 16

[2] C. Labovitz, A. Ahuja, and F. Jahanian, “Experimental study of Internet stability and wide-area network failures,” in Proc. 29th Annual International Symposium on Fault-Tolerant Computing, pp. 278-285, June 1999. 1, 3

[3] A. Ben-Tal and A. Nemirovski, “Robust convex optimization,” Math Oper. Res., vol. 23, no. 4, pp. 769-805, 1998. 3

[4] S. Boyd, and L. Vandenberghe, “Convex Optimization, pp. 318-324, Cambridge University Press, United Kingdom, 2005. 3

[5] J. M. Mulvey, R. J. Vanderbei, S. A. Zenios, “Robust optimization of large-scale systems, Oper. Res. 43, pp. 264-281, 1995. 4

[6] A. L. Soyster, “Convex programming with set-inclusive constraints and applications to inexact linear programming, Oper. Res. 21, pp. 1154-1157, 1973. 4

[7] C. Labovitz, G. R. Malan, and F. Jahanian, “Internet routing instability,” IEEE/ACM Trans. on Networking, vol. 6, pp. 515-528, 1998. 1

[8] Ihsen A. Ouedraogo, “Optimization Models for Robust and Power Efficient Networks,” `http://id.nii.ac.jp/1438/00000789/`, accessed Apr. 2018. 2

[9] S. Jordan, “Traffic Management and Net Neutrality in Wireless Networks,” in IEEE Transactions on Network and Service Management, vol. 8, no. 4, pp. 297-309, December 2011. 2

[10] T. Bauschert, C. Busing, F. D'Andreagiovanni, A. C. A. Koster, M. Kutschka, and U. Steglich, "Network planning under demand uncertainty with robust optimization," in IEEE Communications Magazine, vol. 52, issue 2, pp. 178-185, Feb. 2014. 2

[11] E. Mulyana and U Killat, "Optimizing IP networks for uncertain demands using outbound traffic constraints," In Proc. INOC 2005 (2005) pp. 695-701. 2

[12] D. Mitra, Q. Wang, "Stochastic traffic engineering for demand uncertainty and risk-aware network revenue management," in IEEE/ACM Transactions on Networking, vol. 13, issue 2, pp. 221-233, Apr. 2015. 2

[13] W. Ben-Ameur, H. Kerivin, "Routing of Uncertain Demands," INFORMS, 2001. 2

[14] X. Liu, "Network Optimization with Stochastic Traffic Flows," in International Journal of Network Management, vol. 12, pp.225-234, 2002. 2

[15] A. Altin, B. Fortz, and H. Umit, "Oblivious OSPF routing with weight optimization under polyhedral demand uncertainty," Networks, vol. 60, issue 2, pp. 132-139, Apr. 2012. 2

[16] Y. d'Halluin, P. A. Forsyth, and K. R. Vetzal, "Managing capacity for telecommunications networks under uncertainty," in IEEE/ACM Transactions on Networking, vol. 10, no. 4, pp.579-588, 2002.

[17] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot,"Analysis of link failures in a large IP backbone, " in Proc. 2nd ACM SIGCOM Internet Measurement Workshop, pp. 237-242, Nov. 2002. 2, 7, 16

[18] A. Markopulu, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in Proc. IEEE INFOCOM, pp. 2307-2317, Mar. 2004. 2, 16

[19] S. N. Avci, X. Hu, and E. Ayanoglu, "Recovery from link failures in networks with arbitrary topology via diversity coding," IEEE Global Telecommunications Conference (GLOBECOM 2011), 2011. 2, 3

[20] X. Zhao, B. Zhang, D. Massey, A. Terzis, and L. Zhang, "The Impacts of Link Failure Location on Routing Dynamics: A Formal Analysis," in ACM SIGCOM Asia workshop, 2005. 3

[21] J. P. Vasseur, M. Pickavet, and P. Demeester, "Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS, Elsevier, 2004. 2, 3

[22] J. Subhlok, P. Lieu, B. Lowekamp, "Automatic Node Selection for High Performance Applications on Networks," in ACM SIGPLAN symposium on Principles and practice of parallel programming, pp.163-172, 1999. 3

[23] Y. Zhang, Z, M. Mao, and J. Wang, "A Firewall for Routers: Protecting against Routing Misbehavior," in 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), pp. 20-29, 2007. 3

[24] D. Fang, R. Govindan, J. Heidemann, "An Empirical Study of Router Response to Large BGP Routing Table Load," ACM, Nov. 2002. 3

[25] H. Gredler, W. Goralski, "The Complete IS-IS Routing Protocol," Springer, 2005.

[26] RFX 4593, "by one node to other nodes within the network," `https://tools.ietf.org/html/rfc4593`, 2006. 3

[27] J. Martin, "How to connect to your router to change settings," `http://www.pcadvisor.co.uk/how-to/network-wifi/`, accessed Jan. 2018.

[28] J. T. Moy, "OSPF Version 2. Network Working Group, " 2004. 5

[29] A. Arora, Dijkstra's algorithm, `https://www.slideshare.net/ami_01/`, accessed May 2018. 5

[30] E. Oki, and A. Iwaki, "F-TPR: Fine two-phase IP routing scheme over shortest paths for hose model", IEEE Commun. Letters, Vol.13, No.4, Page: 277-279, Apr. 2009.

[31] CISCO, "Configuring OSPF," `https://www.cisco.com/c/en/us/td/docs/security/asa/asa82/configuration/guide/config/route_ospf.html`, accessed May 2018. 5, 29

[32] L. S. Buriol, M. C. Resende, C. C. Ribeiro, and M. Thorup, "A hybrid genetic algorithm for the weight setting problem in OSPF/IS- IS routing," Networks, 46(1):36-56, June 2005. 6

[33] B. Fortz and M. Thorup, "Optimizing ospf/is-is weights in a changing world," IEEE Journal on Selected Areas in Communications, vol. 20, no. 4, pp. 756-767, May 2002. 6

[34] Reducing link failure and topology change notification times in is-is networks, `https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_isis/configuration/xe-3s/irs-xe-3s-book/irs-fscnt.html`, accessed May 2018. 7

[35] I. M. Kamrul and E. Oki, "Optimization of OSPF link weights to counter network failure," IEICE Trans. on Commun., E94B, (7), pp. 1964-1972, July 2011. 7

[36] R. S. Ranaweera, I. M. Kamrul, and E. Oki, "Preventive start-time optimization of OSPF link weights against link failure for hose model," 18th Asia-Pacific Conference on Communications (APCC 2012), Oct. 2012. 7

[37] R. S. Ranaweera, I. M. Kamrul, and E. Oki, "Preventive start-time optimization of OSPF link weights for hose model," IET Networks, IET Networks, vol. 3, issue. 2, pp. 143-149, June 2014. 7

[38] R.S. Ranaweera, I.A. Ouédraogo and E. Oki, "Network Optimization for Energy Saving Considering Link Failure with Uncertain Traffic Conditions," in IEICE Trans. Commun., vol. E97-B, no. 12, pp. 2729-2738, Dec. 2014. 7, 36

[39] J. Chu and C. Lea, "Optimal Link Weights for IP-Based Networks Supporting Hose-Model VPNs," in IEEE/ACM TRANSACTIONS ON NETWORKING, vol.17, no.3, pp. 778-788, June 2009. 2, 8, 18, 20, 21, 22, 39

[40] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, "A flexible model for resource management in virtual private networks," in ACM SIGCOMM Computer Communication Review, 29, pp. 95-108, 1999. 2, 8

[41] Apache Hadoop, `http://hadoop.apache.org/`, accessed May 2018. 8

[42] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in USENIX Symposium on Operating Systems Design and Implementation, San Francisco, CA, pp. 137-150, Dec. 2004. 8, 50

[43] `https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`, accessed May 2018. 9

[44] Hadoop The Devinitive Guide 4th Edition, Tom White, O'reilly, April 2015. 9, 47, 48, 49

[45] `https://wiki.apache.org/hadoop/topology_rack_awareness_scripts`, accessed May 2018.

[46] `https://docs.openstack.org/sahara/latest/`, accessed Jan. 2018.

[47] `https://issues.apache.org/jira/browse/HADOOP-8468`, accessed Jan. 2018.

[48] J. Tan, X. Meng, and L. Zhang, "Delay Tails in MapReduce Scheduling," in SIGMETRICS Perform. Eval. Rev., vol. 40, no. 1, pp. 5-16, June 2012. 10

[49] J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic, and S. Rao, "Efficient Queue Management for Cluster Scheduling," in Proceedings of the Eleventh European Conference on Computer Systems, pp. 36:1-36:15, 2016. 10

[50] T. Li, J. Tang and J. Xu, "Performance Modeling and Predictive Scheduling for Distributed Stream Data Processing," in IEEE Transactions on Big Data, vol. 2, no. 4, pp. 353-364, Dec. 2016. 10, 62

[51] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," in Proceedings of the 5th European Conference on Computer Systems, pp. 265-278, April 2010. 10

[52] H. Jin, X. Yang, X. H. Sun, and I. Raicu, "ADAPT:Availability-Aware MapReduce Data Placement for Non-dedicated Distributed Computing," in IEEE International Conference on Distributed Computing Systems, Macau, China, pp. 516-525, 2012. 10, 49

[53] W. Dai, I. Ibrahim, M. Bassiouni, "A New Replica Placement Policy for Hadoop Distributed File System," 2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), New York, USA, pp. 262-267, 2016. 10, 49

[54] M. Y. Eltabakh, Y. Tian, F. Ozcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop," Proc. VLDB Endow., vol. 4, no. 9, pp. 575-585, Jun. 2011. 10, 49

[55] Sandvine, "Network Congestion Management: Considerations and Techniques; An Industry Whitepaper," 2015.

[56] E. Oki and A. Awaki, "Load-balanced ip routing scheme based on shortest paths in hose model," in IEEE Transactions on Communications, 58(7), pp. 2088-2096, March 2010. 15, 16

[57] B. Towles and W. J. Dally, "Worst-case traffic for oblivous routing functions," in IEEE Computer Architecture Letters, 1(1), 2002. 16

[58] D. Thaler and C. Hopps, "Multipath Issues in Unicast and Multicast Next-Hop Selection," IETF RFC 2991, Nov. 2000. 17, 31

[59] D. S. Johnson et al., "Optimization by simulated annealing: An experimental evaluation," Operations Research, vol. 37, issure 6, pp. 865-892, Nov.-Dec. 1991.

[60] F. Glover, "Tabu Search - Part 1 and Part 2," ORSA Journal on Computing 1 and 2, 1989. 20

[61] M. Pioro and D. Medhi, "Routing, Flow, and Capacity Design in Communication and Computer Networks," Elsevier, 2004. 23
in IEEE J. Selected Areas in Communications, vol. 14, no. 5, pp. 840-851, June 1996.

[62] `http://www.cs.bu.edu/brite/`, accessed May 2018. 23, 27

[63] K. W. Roth and A. D. Little., "Energy consumption by office and telecommunications equipment in commercial buildings volume I: energy consumption baseline," National Technical Information Service (NTIS), U.S. Department of Commerce, Springfield, VA 22161, NTIS Number: PB2002-101438. 35

[64] M. Zhang et al., "GreenTE: Power-Aware Traffic Engineering," in IEEE International Conference on Network Protocols (ICNP), pp. 21-30, Oct. 2010. 35

[65] J. Guichard et al., "Definitive MPLS Network Designs," Cisco Press, 2005. 35

[66] S. Iyer, S. Bhattacharyya, N. Taft, N. McKeoen and C. Diot, "A measurement based study of load balancing in an IP backbone," May 2002, Sprint ATL technical report, TR02-ATL-051027. 17, 31

[67] P. Zikopoulos and C. Eaton, "Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data", Mcgraw-hill, 2011. 47

[68] I. Polato, R. Re, A. Goldman, and F. Kon, "A Comprehensive View of Hadoop Reseach - A Systematic Literature Review," in Journal of Network and Computer Applications, vol. 46, pp. 1-25, Nov. 2014. 48

[69] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, S. Wu, "Maestro: Replica-Aware Map Scheduling for MapReduce," in 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Canada, pp. 435-442, 2012. 49, 62

[70] H. Jin, X. Yang, X. H. Sun, and I. Raicu, “ADAPT:Availability-Aware MapReduce Data Placement for Non-dedicated Distributed Computing,” in IEEE International Conference on Distributed Computing Systems, Macau, China, pp. 516-525, 2012. 10, 49

[71] W. Dai, I. Ibrahim, M. Bassiouni, “A New Replica Placement Policy for Hadoop Distributed File System,” in IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), New York, USA, pp. 262-267, 2016. 10, 49

[72] M. Y. Eltabakh, Y. Tian, F. Ozcan, R. Gemulla, A. Krettek, and J. McPherson, “CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop,” Proc. VLDB Endow., vol. 4, no. 9, pp. 575-585, Jun. 2011. 10, 49

[73] R.S. Ranaweera, E. Oki, and N. Kitsuwan, “Delay Distribution Based Remote Data Fetch Scheme for Hadoop Clusters in Public Cloud,” in IEICE Trans. Commun., vol. E102-B, No.8, Aug. 2019. 54, 55

[74] R.S. Ranaweera, E. Oki, and N. Kitsuwan, “Non-local Data Fetch Scheme Based on Delay Distribution for Hadoop Clusters in Public Cloud,” The 4th IEEE International Conference on High Performance and Smart Computing (IEEE HPSC 2018), May 2018. 54, 55

[75] `https://spark.apache.org/`, accessed May 2018. 48

[76] `https://tez.apache.org/`, accessed May 2018. 48

[77] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, “Quincy: Fair Scheduling for Distributed Computing Clusters,” in Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, ACM, pp. 261-276, 2009 51, 62

[78] C. E. Leiserson, “Fat-trees: Universal Networks for Hardware-efficient Supercomputing,” in IEEE Trans. Comput., vol. 34, no. 10, pp. 892-901, Oct. 1985. 51

[79] R. Carter and M. Crovella, “Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks,” in Technical Report, Boston University, Boston, USA, 1996. 53

[80] H. Miura and M. Yamamoto, “Content routing with network support using passive measurement in content distribution networks,” in Proceedings. Eleventh International Conference on Computer Communications and Networks, Miami, FL, USA, pp. 96-101, 2002. 54

[81] T. Mori, T. Hirata, and M. Yamamoto, “Content-oriented probabilistic routing with measured RTT,” in 2016 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR 2016), Stevenson, WA, pp. 1-6, 2016. 54

[82] Y. Han, M. Kim, and H. Park, “A novel server selection method to achieve delay-based fairness in the server palm,” in IEEE Communications Letters, vol. 13, no. 11, pp. 868-870, November 2009. 54

[83] V. N. Padmanabhan, J. C. Mogul, “Improving HTTP latency,” in Computer Networks and ISDN Systems, vol. 28, issues 1-2, pp. 25-35, 1995. 54

[84] D. Sanghi, P. Jalote, P. Agarwal, “Using Proximity Information for Load Balancing in Geographically Distributed Web Server Systems,” in Information and Communication Technology, EurAsia-ICT 2002, Lecture Notes in Computer Science, vol. 2510, Springer, Berlin, Heidelberg, 2002. 54

[85] S. Salsano, F. Patriarca, F. L. Presti, P. L. Ventre and V. M. Gentile, “Accurate and Efficient Measurements of IP Level Performance to Drive Interface Selection in Heterogeneous Wireless Networks,” in IEEE Transactions on Mobile Computing, vol. 17, no. 10, pp. 2223-2235, 1 Oct. 2018. 54

[86] M. E. Crovella and R. L. Carter, “Dynamic server selection in the internet,” in Proc. of the 3rd IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS 95), June 1995. 54

[87] C. C. Holt, “Forecasting seasonals and trends by exponentially weighted moving averages,” (O.N.R. Memorandum No. 52), Carnegie Institute of Technology, Pittsburgh USA, 1957. 55

[88] R. G. Brown, “Statistical forecasting for inventory control,” McGraw/Hill, 1959. 55

[89] “Forecasting Principles and Practice,” `https://otexts.org/fpp2/expsmooth.html`, accessed November 2018. 55

[90] B. Huffaker, M. Fomenkov, D. Plummer, D. Moore, and K. Claffy, “Distance Metrics in the Internet,” in IEEE International Telecommunications Symposium (ITS), pp. 200-202, 2002.

[91] `https://aws.amazon.com/ec2/`, accessed January 2018. 59, 60

[92] `https://www.cloudera.com/documentation/enterprise/release-notes/topics/cdh_vd_cdh_package_tarball.html`, accessed Sep. 2017. 60

[93] `https://hadoop.apache.org/docs/r2.6.5/hadoop-yarn/hadoop-yarn-site/FairScheduler.html`, accessed Sep. 2017. 60

[94] S. Kaptchouang, I. A. Ouedraogo, and E. Oki, “Strengthened Preventive Start-Time Optimisation that Reduces Congestion Ratio under No Failure Scenario,” The Journal of Engineering, vol. 2016, issue 12, pp.441-332, 2016.

[95] G. B. Dantzig, Linear programming under uncertainty, Mgmt. Sci., 197-206, 1955. 71

[96] `https://www.opennetworking.org/sdn-definition/`, accessed Sep. 2017. 71

Publications

List of Publications related to the thesis

Journal Papers

1. R.S. Ranaweera, E. Oki, and N. Kitsuwan, "Delay Distribution Based Remote Data Fetch Scheme for Hadoop Clusters in Public Cloud," in IEICE Trans. Commun., vol. E102-B, No.8, Aug. 2019 (paper accepted).

2. R.S. Ranaweera, I.A. Ouédraogo and E. Oki, "Network Optimization for Energy Saving Considering Link Failure with Uncertain Traffic Conditions," in IEICE Trans. Commun., vol. E97-B, no. 12, pp. 2729-2738, Dec. 2014 (paper).

3. R.S. Ranaweera, I.M. Kamrul, and E. Oki, "Preventive start-time optimisation of open shortest path first link weights for hose model," IET Networks, vol. 3, issue. 2, pp. 143-149, June 2014 (paper).

International Conference Papers

1. R.S. Ranaweera, E. Oki, and N. Kitsuwan, "Non-local Data Fetch Scheme Based on Delay Distribution for Hadoop Clusters in Public Cloud," The 4th IEEE International Conference on High Performance and Smart Computing (IEEE HPSC 2018), May 2018.

2. R.S. Ranaweera, I.M. Kamrul, and E. Oki, "Preventive start-time optimization of OSPF link weights against link failure for hose model," 18th Asia-Pacific Conference on Communications (APCC 2012), Oct. 2012.

National Conference Papers

1. R.S. Ranaweera, I.A. Ouédraogo, and E. Oki, “Performance Evaluation of Preventive Start-time Optimization for Energy Saving of Hose Model against Link Failure,” IEICE Tech. Rep., vol. 113, no. 472, NS2013-236, pp. 343-348, March 2014.

2. R.S. Ranaweera, I.M. Kamrul, and E. Oki, “Performance Evaluation of Preventive Start-Time Routing Optimization for Hose Model,” IEICE Tech. Rep, vol. 113, no. 91, PN2013-4, pp. 19-24, May. 2013.