Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Telecommunication Engineering

Master's Thesis

Optimization of Network Architecture for Hadoop Mapreduce

May 10, 2015

Author: Bc. Zdeněk Kouba
Supervisor: Dr. Lukáš Kencl

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra telekomunikační techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Kouba Zdeněk**

Studijní program: Komunikace, multimédia a elektronika
Obor: Sítě elektronických komunikací

Název tématu: **Optimalizace síťové architektury pro Hadoop Mapreduce**

Pokyny pro vypracování:

Práce se zabývá studiem vlivu síťové topologie datového centra na rychlost výpočtu úloh Hadoop / Mapreduce. Budou zkoumány různé topologie a studovány možnosti dynamického přizpůsobení topologie při nerovnoměrné zátěži algoritmu způsobené různým statistickým rozdělením úloh, např. dle mocninného zákona. Vliv topologií bude studován za použití simulátoru CloudSim a dalších nástrojů.

Seznam odborné literatury:

[1] Apache hadoop. http://hadoop.apache.org/.

[2] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In Computer and Information Technology (CIT), 2010 IEEE 10th
International Conference on.

[3] Yuanquan Fan, Weiguo Wu, Haijun Cao, Huo Zhu, Wei Wei, and Pengfei Zheng. Lbvp: A load balance algorithm based on virtual partition in hadoop cluster. In Cloud Computing Congress (APCloudCC), 2012 IEEE Asia Pacific.

[4] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In Proceedings of the
5th European conference on Computer systems, 2010.

Vedoucí: Dr. Lukáš Kencl

Platnost zadání: do konce letního semestru 2015/2016

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.
-4-

prof. Ing. Boris Šimák, CSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 2. 2. 2015

Aknowledgements

I would like to express my gratitude for help and guidance throughout the course of the last year to my advisor, Dr. Lukáš Kencl. I'd also like to thank the Department of Telecommunications at CTU and especially the head of this department prof. Ing. Boris Šimák, CSc. for generously providing me with means and excellent environment for my research and studies. And last, but definitely not least, I want to express my deep gratitude to M.SE. Dazhuo Li, Phd. for mentoring me and introducing me to the field of distributed computing.

Anotace

This thesis studies the influence of datacenter network topology on performance of distributed computing tasks following the MapReduce model. The first part of this thesis gives an overview of MapReduce in general, including an example usecase, and state of the art technologies and publications in this field. The influence of datacenter topology on MapReduce performance is evaluated through a series of simulations. Design of those simulation scenarios as well as choice of simulator and implementation of a network topology simulation module extending that simulator and improvement of this simulator's scheduling algorithm are also described in this thesis. The final part of this thesis presents results of the conducted simulations and conclusion of this work.

Index Terms:
cloud, Hadoop, MapReduce, network topology, simulation, CloudSim

Anotace

Tato diplomová práce studuje vliv síťové topologie datacentra na dobu trvání distribuovaných výpočtů typu MapReduce. Úvodní část této práce poskytuje obecný přehled technologie MapReduce včetně jednoduchého příkladu jejího využití a zabývá se také nejnovějšími technologiemi a publikacemi v této oblasti. Vliv topologie datacentra na výkonnost MapReduce je studován pomocí simulací. Návrh simulačních scénářů, volba simulátoru i rozšíření zvoleného simulátoru o modul pro simulaci síťové topologie a úprava stávajícího algoritmu plánovače úloh implementovaného ve zvoleném simulátoru jsou přemětem následující části této diplomové práce. V závěru jsou prezentovány a vyhodnoceny výsledky simulací.

Klíčová slova:
cloud, Hadoop, MapReduce, síťová topologie, simulace, CloudSim

Contents

1 Introduction **21**
- 1.1 MapReduce . . . 22
  - 1.1.1 MapReduce example usecase . . . 23

2 Related Work **27**
- 2.1 Scheduling optimization . . . 27
- 2.2 Shuffle/Merge phase optimization . . . 28
- 2.3 Datacenter architecture design . . . 28
- 2.4 MapReduce Simulation . . . 30

3 Methodology **33**

4 Simulator Implementation **35**
- 4.1 Integration with CloudSim . . . 35
- 4.2 MapReduce Job Finish Time Calculation . . . 37
- 4.3 Topology Representation . . . 39
  - 4.3.1 Routing . . . 41
  - 4.3.2 Throughput Allocation . . . 42
- 4.4 Selection of Optimal Scheduling Plan . . . 42

5 CloudDynTop simulator deployment **45**
- 5.1 CloudDynTop input arguments and files . . . 45
  - 5.1.1 Utility scripts . . . 47
- 5.2 CloudDynTop output files . . . 48

6 Simulation setup **51**
- 6.1 Simulated Topologies . . . 51
  - 6.1.1 Hierarchical Topology . . . 52
  - 6.1.2 BCube Topology . . . 52
  - 6.1.3 MapReduce Topology . . . 53
  - 6.1.4 $BCube_1$ Topology . . . 53
  - 6.1.5 CamCube Topology . . . 53
- 6.2 Simulated Workloads . . . 54

7 Results **57**
7.1 MapReduce Performance 57
7.2 Simulator Performance 60

8 Conclusion **61**

A Contents of attached CD **67**

List of Figures

1.1 Illustration of topology suffering from bottleneck. 22
1.2 Illustration of well performing topology. 22
1.3 Illustration of data flow in a MapReduce job 25

2.1 DCell topology illustration (taken from [15]) 30

3.1 Workflow of presented simulation setup 33

4.1 CloudDynTop component diagram 36
4.2 Sequence Diagram - Interactions between PredictionEngine and CloudDynTop 36
4.3 Sequence Diagram - Interactions between MapReduceEngine and CloudDynTop 37
4.4 Class Diagram - Process, Flow, Computation 38
4.5 Class diagram of topology representation 40
4.6 Available share versus fair share throughput allocation 42

5.1 Example of job description specified in YAML format 47
5.2 Example of output of CloudDynTop providing process time statistics. 50

6.1 Illustration of the 3-Tier hierarchical topology 51
6.2 Illustration of BCube topology 52
6.3 Illustration of 3D Torus topology used in CamCube network stack 53
6.4 Simulated data distributions 55

7.1 Job finish times for exponential data distributions 58
7.2 Job finish times for normal data distributions and job sizes 58
7.3 Job finish times for uniform data distributions 59
7.4 Comparison of job finish times when available throughput share allocation resp. 59
7.5 Simulator's running time. 60

List of Tables

2.1 Comparison of MapReduce simulators 31

5.1 Command line arguments of CloudDynTop 46
5.2 Topology types supported by CloudDynTop 47

6.1 Parameters of BCube topology 52

List of abbreviations

3D Torus	Datacenter topology (see 6.1.5)
3-Tier topology	Hierarchical network topology consisting of 3 layers - edge, aggregation and core
BCube	Datacenter topology (see [14])
BRITE	Network topology generation, serialization and deserialization framework
CamCube	Datacenter network stack (see [5])
CloudDynTop	Network topology simulation module implemented as part of this thesis
CloudSim	Cloud simulator (see [9])
CloudSimEx MapReduce	MapReduce simulator (see [2])
CPU	Central Processing Unit
DCell	Datacenter topology (see [15])
EIGRP	Enhanced Interior Gateway Routing Protocol
FatTree	Datacenter topology (see [21])
GridSim	Framework for simulating grid, cluster and peer-to-peer algorithms (see [8])
Hadoop	Family of projects providing a big data platform (see [1])
HDD	Hard Disk Drive
JSON	Standard for object serialization (JavaScript Object Notation)
Mapper	Map task of a MapReduce job
MapReduce	Distributed computing model (see [11])
Matlab	a high-level technical computing language and interactive environment
MB	Megabyte
MEMS	Microelectromechanical systems
MIPS	Million Instructions per Second
MRPerf	MapReduce simulator (see [26])
MRSG	MapReduce simulator (see [18])
MRSim	MapReduce simulator (see [16])
NS2	Network topology simulator
Reducer	Reduce task of a MapReduce job

SDN	Software Defined Networks
SimGrid	Toolkit for simulating application scheduling (see [10])
SimJava	Discrete event simulation framework (see [3])
SkewTune	Algorithm for balancing computational load skew in MapReduce, proposed in [20]
VM	Virtual Machine
YAML	Standard for object serialization (YAML Aint Markup Language)

Preface

The motivation of this work was to study dependency of performance of distributed computing tasks following the MapReduce model on workload type and network topology through simulations. Acquired knowledge can be used in a future work to design a system for dynamically adjusting network topology to the current workload in order to optimize MapReduce performance.

In this thesis, we devoted significant efforts to evaluate the state of the art publications and technologies in the field of optimization of MapReduce and datacenter architecture design and simulation. Based on this research we have chosen a simulator (CloudSimEx MapReduce) and designed a set of simulation scenarios to gain understanding of the relation between workload type, network topology and the resulting performance of MapReduce.

During our work towards conducting these simulations we had to tackle several issues imposed by the chosen simulator. After examining its source code it became apparent that several of the features of the simulator have to be used individually, but cannot be combined as was necessary for our research. Moreover, we have identified suboptimal behavior of its scheduling algorithm that made our simulations run unacceptably long. To overcome these problems, we have implemented an extension module to this simulator called CloudDynTop, that addresses all of these issues. The CloudDynTop module is also presented in this thesis.

In the end we have been able to conduct the proposed simulations and we present the derived conclusions in chapter 7.

Chapter 1

Introduction

Datacenter design is a broad and complex topic. It's many aspects include design of network topology, routing services, monitoring and resource management services and many others. Moreover the importance of this field still increases, one of the main driving factors being the "Big Data" trend. New data processing technologies have enabled for example machine learning systems to digest datasets larger than ever. And datasets are growing at an increasing speed, partly driven by the emerging era of Internet of Things.

One of the most commonly used data processing technologies is *MapReduce*. A closer description of this programming model is given in the following section. The most popular implementation of MapReduce is part of the *Hadoop* [1] project family. MapReduce system's performance is significantly influenced by the underlying network, due to the volume of network traffic that this framework generates. The most significant portion of this traffic comes from the shuffle/merge phase of MapReduce when output of the first phase is transmitted from Map nodes to Reduce nodes. The duration of transmission of this data determines when the Reduce phase can start (due to the serialization barrier of MapReduce). Since these systems inherently produce often very large volume of network traffic, network topology has a great impact on their performance. Example scenarios are given on figures 1.1 and 1.2. Figure 1.1 illustrates a scenario, where Map tasks are in a different network segment than Reduce tasks and the network connecting these segments becomes the bottleneck for the data flow during the shuffle/merge phase. Data is being transmitted from each Mapper to all Reducers and each of these flows is passing through the links between SW1 and SW2, resp. SW2 and SW3. These links limit the throughput of the flows, leaving links connecting the Mapper and Reducers to SW1, resp. SW3 underutilized. On the other hand, Figure 1.2 presents a topology, where each data flow has its own dedicated link. Every connection is utilized to its full capacity.

Another important factor in datacenter architecture are software-defined networks (SDN). This powerful technology offers great control over network and traffic passing through it. Utilizing the capabilities of these systems in combination with a monitoring system keeping track of workload being processed by the datacenter would allow for dynamic changes to network topology with regards to current workload of the MapReduce system, thus creating time and energy efficient environment.

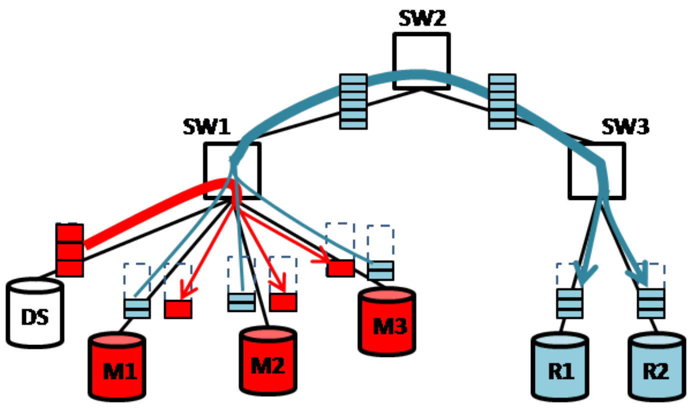

Figure 1.1: Illustration of topology suffering from bottleneck. The dotted rectangles represent capacity of the link. Empty space in a dotted rectangle illustrates wasted resources (here throughput). Red rectangles represent flows from the data source (DS) to Mapper nodes. Their height represents throughput allocated for them. Similarly, the blue rectangles represent flows from Mapper nodes to Reducer nodes. This figure illustrates impact of a bottleneck links between the data source node and switch SW1 and the links SW1 -> SW2 and SW2 -> SW3 on flows from the data source to the Mapper nodes, and from Mapper nodes to Reducer nodes respectively.

1.1 MapReduce

MapReduce is a programming model aimed at distributed processing of large amounts of data. It's been proposed in the white paper [11]. Publication of this paper is often considered to have triggered the current "big data" trend and MapReduce became one of key components of several in industry heavily used systems, such as Apache Hadoop [1].

The MapReduce model defines 3 phases:

1. Map
2. Shuffle/Merge
3. Reduce

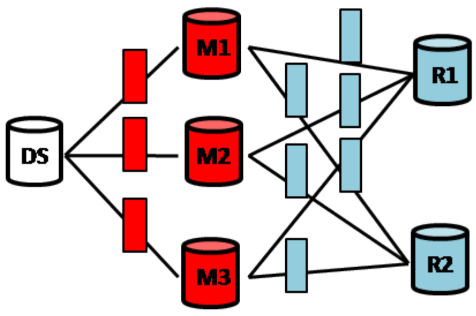

Figure 1.2: Illustration of well performing topology. Each flow has its own dedicated link, therefore there are no bottleneck constraints.

A MapReduce job consists of M Map tasks (Mappers) and R Reduce tasks (Reducers). Each Mapper is assigned a portion of the job's input data (typically distributed among Mappers uniformly). The data is parsed as key-value pairs and Mappers process them one pair at a time. Mappers produce on their output new key-value pairs of intermediary data. If several pairs should have the same key, their values are merged into a list and sent to output under that key. Each Mapper sorts its output by keys and organizes it into R partitions. The index r of the partition to which a key-value pair belongs is calculated using the following formula:

$$r = hash(key)\ mod\ R \tag{1.1}$$

where $hash(x)$ is a function computing hash of the intermediary keys and R is the number of Reducers.

When a Mapper has processed all of its input, each of the created partitions is sent to the rth Reducer. This is part of the Shuffle/Merge phase. At Reducer side, all received partitions are sorted by keys. Each Reducer can start processing its input only after all Mappers have finished (end of the Map phase) and its input has been sorted (end of the Shuffle/Merge phase and start of the Reduce phase). Similarly to Mappers, Reducers process their input one key-value pair at a time and they produce new key-value pairs. Those pairs are the final output of that MapReduce job. An illustration of data flow in a MapReduce job can be found on Figure 1.3.

If the function implemented by Reducers is commutative, the MapReduce job can be configured with an optional step called Combining. If this is the case, the finished output partitions of each Mapper are processed by the reduce function locally on the Mapper's host before they are sent over the network to the Reducer's host. This optimization can significantly reduce the total volume of intermediary data, but it is not always possible to utilize it.

1.1.1 MapReduce example usecase

In this subsection, we provide an illustrative usecase to show how the MapReduce model works. This example will describe the process of obtaining a sorted list of words from a large data set.

First of all, the input data set is divided into roughly equal-sized input splits. Each Mapper receives one of those splits and starts processing it. In this case, Mappers could be configured to parse the input into key-value pairs, where key could be offset within the input split (number of character already read) and value would be a word at that offset. This is the default configuration of Hadoop MapReduce.

Mappers process their input one key-value pair at a time. For each offset-word pair, every Mapper outputs new key-value pair, where the word becomes the key. Value of this intermediary key-value pair is not important in this particular case. When a Mapper sends a key-value pair to its output, the key value pair is inserted into one of the Mapper's R output partitions, which one is determined using formula 1.1. If the partition already contains the key of the pair being inserted, value of this pair is simply appended to an array of values associated with that key. The partitions keep the keys in a sorted order. As a result, each

Mapper will create a sorted list of words (each word appearing at most once). These sorted list produced by the Mappers need to be merged into a single list - which is the task of the Reducer. Hence, there will be only one Reducer in this case and therefore each Mapper will create only one output partition.

At this point the Combining phase would typically follow in some real-life scenarios. However, this particular usecase has no use for the Combining phase.

Once a Mapper is done with processing all of its input (including the Combining step if applicable), it sends the resulting partition to the Reducer (generally when there are multiple Reducers, the ith partition is sent to the ith Reducer). The Reducer organizes the received key-value pairs into an input partition, sorting the keys and merging values of duplicate keys into arrays similarly to how Mappers inserted their output key-value pairs into their output partitions. After all Mappers have finished processing their input and sending their output, the Reducer can start processing its input partition, one key-value pair at a time. In this case the Reducer will simply take the key of the key-value pair and send it to its output. This will create a sorted list of words, each word appearing only once on the list.

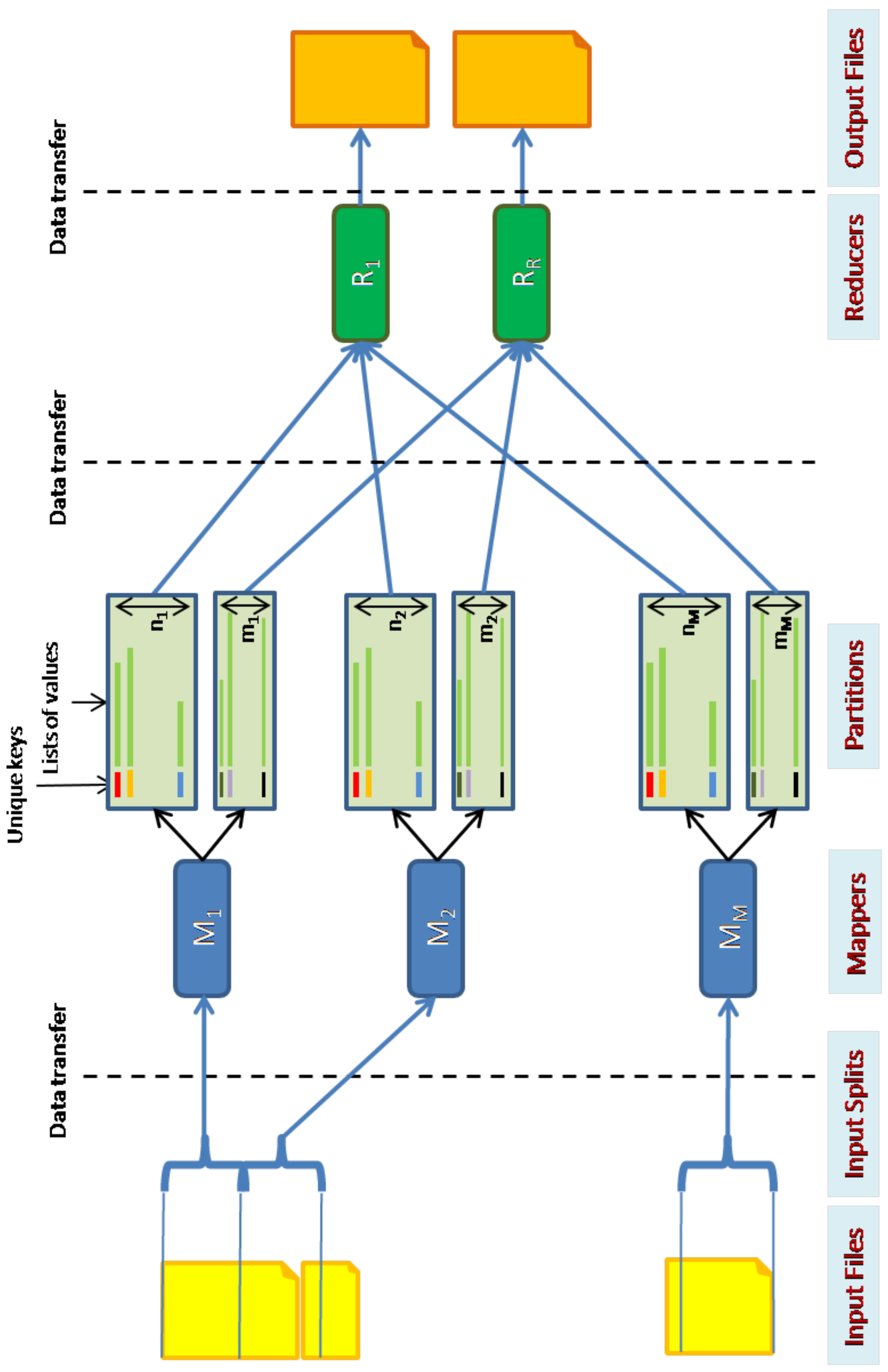

Figure 1.3: Illustration of data flow in a MapReduce job. In this example, the MapReduce job consists of M Mappers and R Reducers. Output of each Mapper is organized into R partitions (although only 2 are illustrated here). Points where data is being transmitted over network are highlighted by vertical dotted lines.

Chapter 2

Related Work

Performance of MapReduce comprises of several factors - among others task scheduling, skew mitigation, struggler detection, topology optimization. The following sections of this chapter present some of the research and publications studying those fields that relate to this thesis.

2.1 Scheduling optimization

In task scheduling, one of the main difficulties is the trade-off between data locality and fairness, as is pointed out in paper [30]. Here locality in scheduling means placing the tasks on machines that have a copy of the data required by the task on their local file system. But these machines aren't available all the time, thus to achieve complete locality, the scheduler would make tasks wait for a machine with a copy of the task's input data, even though there might be other machines in the datacenter available. In extreme scenarios the waiting time might be even higher than time needed to transfer the data to another machine. On the other hand, the data transfer time can often be higher than time spent waiting for a data-local machine to free up. Fairness in context of task scheduling refers to servicing all tasks with best effort, providing them with a fair share of the datacenter's resources as soon as they become available.

The paper [30] discusses the following approaches a) waiting for resources to free up, b) killing tasks in order to free up resources. Proposed algorithm is called *Delay Scheduler*. This scheduler waits up to a configurable deadline for a task slot to free up on a data-local node. If such task slot does become available before the time-out a waiting data-local task is scheduled on that node. If all data-local task slots stay occupied during the whole time-out period one of the waiting tasks is scheduled on any node with a free task slot, regardless of data-locality.

Bandwidth-aware scheduling [22] tackles the locality-fairness trade-off too, but using a a different approach. The proposed scheduler makes decisions based on current conditions in the network. It utilizes information about available bandwidth on links in the topology retrieved from an SDN controller and information about available idle time of individual nodes in the cluster to estimate, whether to wait for a data-local node or not.

A general approach to the question of VM placement with regards to location of data is discussed in [6]. Authors present an algorithm for VM placement minimizing several aspects of data access latency. Some of the discussed optimization objectives are minimization of total/average data access time, or minimization of the worst data access time among all access times in that scenario etc. A series of simulations benchmarking the proposed algorithm is also presented.

2.2 Shuffle/Merge phase optimization

The term skew in context of MapReduce translates to uneven distribution of input data and/or computational load among tasks. An overview and some examples of observed skew in MapReduce is presented in [19]. The issue of skew in computational load is also the focus of paper [20], among others. In this paper, authors propose *SkewTune* - an algorithm for detection of tasks running extraordinarily long, called *strugglers*. When such a task is detected, SkewTune terminates it, takes its input that hasn't been processed yet and redistributes it among a set of new tasks.

A big focus point for optimization efforts in MapReduce is the shuffle/merge phase. It's the process of sorting output of Mapper tasks, organizing it into partitions associated with individual Reduce tasks and transferring it to the nodes where respective tasks are running. One of the paper focusing on this topic is [29]. In this paper, authors point out that current implementation of Hadoop MapReduce leads to repetitive copying/merging of data. They propose an algorithm to minimize volume of data transfered over network during the shuffle/merge phase. Instead of waiting for all Map tasks to finish, Reducers fetch small header files describing range of keys produced by individual finished Mappers. Those headers are used to construct a priority queue (concurrently with map phase) which serves to merge all Map Output Files at once later on.

The paper [12] is also focused on optimization of the shuffle/merge phase of MapReduce. The approach taken here is to insert a load balancer into the data flow between Mappers and Reducers. The proposed load balancer takes partitions of intermediary data created by Mappers and repartitions them into more evenly distributed partitions before passing the data to Reducers. An implementation of this system is also presented in that paper, as well as results of a benchmark of the proposed system.

2.3 Datacenter architecture design

The papers discussed so far were focusing on management and utilization of a datacenter, but design of datacenter architecture is also an important topic and in fact a closer one to the focus of this thesis.

A datacenter architecture based on optical MEMS switches combined with SDN is presented in [28]. The optical switches compose network connecting racks in a datacenter. Physical connectivity and forwarding rules are managed by the SDN controller. The SDN controller communicates with application masters and resource managers in the datacenter making it application-aware. This becomes an advantage for example during task scheduling or when handling various traffic patterns.

This thesis mainly focuses on network topology evaluation. One of the studied topologies is *BCube*, which was presented in [14]. The topology itself will be described in more detail in later section of this thesis (6.1.2). The paper [14] also proposes an addressing scheme for servers and switches in the BCube topology and a routing algorithm based on this addressing scheme.

Another paper focusing on datacenter architecture design is [5]. This paper presents a network stack called *CamCube* based on (but not limited to) 3D Torus topology. This architecture doesn't require any switches, since all routing is done by the servers. One of the main advantages of this approach is the option for custom implementations of different routing algorithms. CamCube offers to applications running on top of it a routing service allowing each application to register its own routing algorithm implementation. Other core services provided by CamCube include VM image distribution, cache services and aggregation services. The 3D Torus topology is described in more detail in section 6.1.5 of this thesis, as it is one of the simulated topologies.

The *DCell* topology has been presented in [15]. Similarly to BCube, DCell is a recursively defined topology. Figure 2.1 provides an illustration of this topology. It is design for extremely large datacenters, consisting of millions of servers. Since common routing algorithms aren't suitable for such conditions, the paper also presents several routing algorithms for this topology.

The *FatTree* topology has been described in paper [21]. It is a tree topology, where processing nodes are located at the leaves and internal nodes of the tree are switches. The key feature of a FatTree topology is that the links closer to the root of the tree have higher capacity that the links closer to the leaves.

Since the DCell and FatTree topology has been subject to previous studies of MapReduce performance, it is not included in simulation scenarios of this thesis.

Design of datacenter topology doesn't focus just on performance, but also on efficiency, as is the case elaborated in [17]. Architecture called *ElasticTree* presented in this paper is designed to increase energy efficiency while providing reasonable performance. This is achieved by utilizing an SDN controller, that realizes decisions of an optimizer (proposed in that paper) determining which links and devices need to be powered on to meet traffic demand constraints and which can be powered off to save energy.

Simulations of datacenter application performance on different network topologies has been presented for example in [7]. This paper evaluates in detail throughput and delay in DCell, FatTree and *3-Tier hierarchical* topology (a tree topology consisting of 3 layers of switches and hosts at the leaves) given a typical traffic in a datacenter (not just MapReduce). The simulated traffic patterns were based on logs from several existing datacenters.

Performance of MapReduce specifically with respect to different topologies has been the focus of [27]. Topologies simulated in this paper were star (hosts are connected to a central "hub" node), double rack (essentially two star topologies with a connection between the central nodes), tree and DCell. The paper also presented a simulator called *MRPerf* and evaluated its performance during those simulations.

Research focused on optimizing effectivity of private clouds is also conducted at Department of Cybernetics of Czech Technical University. In the paper [25], the authors present a setup consisting of *Eucaliptus* private cloud system and queue engine *Cloud Gunther*. This setup integrates control of VM provisioning within the job scheduler, thus allowing for maximal utilization of the cloud's resources.

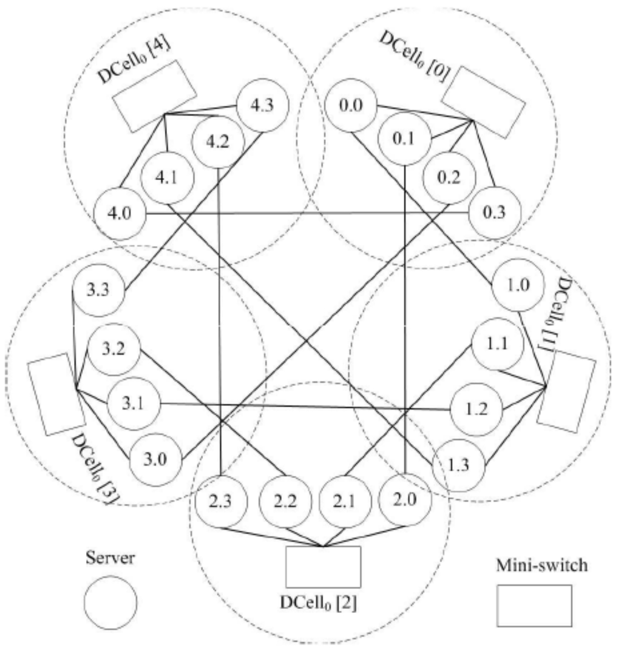

Figure 2.1: Illustration of DCell topology as presented in [15]

2.4 MapReduce Simulation

Since simulation of MapReduce is the core part of this thesis, this section presents comparison of several simulators that were considered as candidates for running our simulations. Table 2.1 summarizes the following key features (rows) of individual simulators (columns):

- Network Sim. Engine - the network topology simulation engine used by the respective simulator
- Language - programming language used for implementation of the respective simulator
- Scalability - how well does the simulator perform with increasing simulation scale
- MapReduce detail - the level of detail to which the simulator simulates the MapReduce framework
- Skew Support - whether the simulator supports simulation of uneven distribution of intermediary data

The *CloudSim* simulator has been presented in [9]. Its architecture follows a layered design composing of i) simulation core, ii) CloudSim and iii) user code layer. Initial releases utilized *SimJava* - a discrete event simulation framework, but later releases removed it to allow for some advanced features not supported by SimJava. The user code layer provides interfaces for specification of scheduling policies, cloud scenarios, application configurations etc. making CloudSim very flexible.

Table 2.1: Comparison of MapReduce simulators

	CloudSim/ CloudSimEx	**MRPerf**	**MRSim**	**MRSG**
Network Sim. Engine	CloudSim	NS2	GridSim	SimGrid
Language	Java	C++, tcl, python	Java	C
Scalability	great	good	poor	great
MapReduce detail	low	low	great	low
Skew Support	yes	no	unknown	yes

The MRPerf simulator presented in [26] has been designed as an accurate simulator of Hadoop. The simulator utilizes *NS2* for network topology simulation. It provides simulation at sub-phase level and simulates intra- and inter- rack communication and even statistics about node usage, such as processor and disk I/O time. A downside of this simulator is that it doesn't support simulation of unbalanced distribution of intermediary data and assumes that the data is distributed uniformly.

The paper [16] presents a discrete event based simulator for Hadoop implementation of MapReduce called *MRSim*. This simulator highly utilizes Object oriented programming, as CPU, HDD or Network interfaces are designed to be basic blocks and can be grouped into server objects. For simulation of network topology MRSim utilizes *GridSim*, rest of the systems is modeled using SimJava discrete event engine. GridSim allows for packet-level simulation of network. It supports space and time shared allocation policies and allows for representing a cluster as a single entity.

Another MapReduce simulator is *MRSG*, prosented in [18]. Unlike MRSim, this simulator is based on *SimGrid* - a simulation framework for evaluating cluster, grid and peer-to-peer algorithms. SimGrid's network model allows form faster simulation times than packet-based GridSim and thanks to that MRSG scales better at cost of lower network simulation accuracy.

Chapter 3

Methodology

This thesis focuses on the effect of network topology on performance of different workload types of MapReduce distributed computing model. To study this problematic, several simulation scenarios have been proposed. These simulations cover cases of different MapReduce job dimensions (numbers of Mappers and Reducers), intermediary data distributions and network topologies. Performance of MapReduce in individual scenarios is evaluated based on job finish times. In each scenario a single MapReduce job is launched in the simulated cluster. Nodes in the cluster are configured with only one task slot and the scheduler only assigns one task to each machine (as opposed to one machine running several tasks sequentially). These constraints were introduce in order to increase performance of the simulator, but more importantly to reduce effect of other aspects of MapReduce performance than the impact of network topology. With that in mind, the simulated jobs were configured with relatively small amount of computational workload compared to the configured volume of data. The simulation setup is described in more detail in Chapter 6.

The simulator chosen to run the simulations is the extension of CloudSim [9] called CloudSimEx MapReduce [2]. It has been chosen for its flexibility, level of detail, as well as

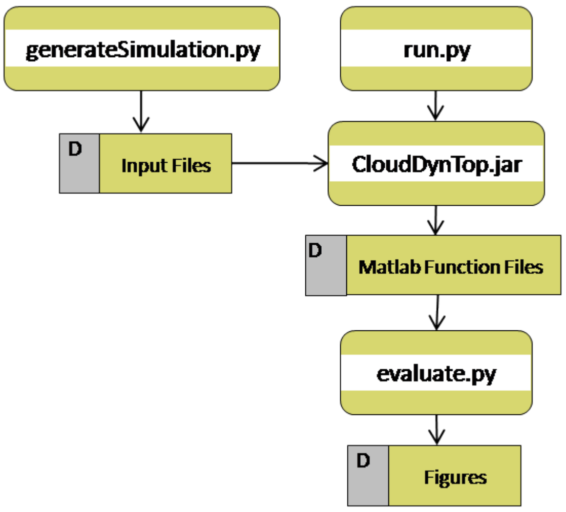

Figure 3.1: Data flow of presented simulation setup

ease of implementation and integration of new modules. Downside of this simulator is the insufficient accuracy of network traffic simulation. To address this issue, a network topology simulation module, called *CloudDynTop*, has also been implemented as an extension of CloudSimEx MapReduce as part of this thesis. This module is discussed in more detail in Chapter 4. One of the problems tackled by CloudDynTop was method of distributing throughput of links among flows passing through them. Chapter 4 discusses two approaches - fair share allocation and available share allocation. Chapter 7 gives evaluation and comparison of these two algorithms (among other results).

Figure 3.1 illustrates the workflow of the presented simulation setup. The CloudSim simulator (process *CloudDynTop.jar* on the diagram) takes input in form of *YAML* files defining dimensions of the simulated jobs, distribution of intermediary data, volume of Mappers' input, computational load on Mappers and Reducers or submission time of the simulated jobs. YAML [4] is a serialization standard, similar to its wider spread peer JSON. It's been designed to provide more user-friendly access to serialized object editing. CloudSimEx MapReduce utilized YAML for job configuration through serialization/deserialization of *Experiment* (represents a set of jobs) and *Job* (represents a single job) classes. To automate creation of these files, utility script *generateSimulation.py* described in Chapter 5 has been implemented. The implemented module CloudDynTop also needs some external configuration (such as type of topology, number of nodes, throughput allocation method etc.) - in this case in form of command line arguments. These parameters are defined in script *run.py* which launches CloudDynTop passing it those arguments.

The output of the simulation are Matlab functions (also described in more detail in Chapter 5), which are then processed in Matlab into figures presented in Chapter 7.

Chapter 4

Simulator Implementation

This chapter discusses the details of the implemented CloudDynTop module. Section 4.1 describes dependencies between CloudSim, CloudSimEx MapReduce and CloudDynTop. Section 4.3 presents algorithms used by CloudDynTop to route flows through the topology, assign throughput shares to flows and calculate job finish time. Scheduling algorithm implemented in CloudSimEx MapReduce and its improvement implemented during development of CloudDynTop is discussed in section 4.4.

4.1 Integration with CloudSim

To run the simulations, the CloudSim simulator has been selected, more specifically the MapReduce module of CloudSimEx project, which serves as an incubator for new features of the main CloudSim project. We have chosen this simulator, because of its good support of simulating skew in distribution of intermediary data, flexibility and ease of implementation and integration of new modules, more over it seems to be popular and well accepted in the datacenter engineering community. However later on the downside of this simulator came up. The problem is how it simulates network topology.

CloudSim is mostly aimed at simulation of multi-datacenter setups. Although it does support simulation of intra-datacenter topology in its org.cloudbus.cloudsim.network.datacenter package, it is only capable of simulating 3-tier hierarchical topology. Unfortunately, CloudSimEx MapReduce is incompatible with this feature. The root of this incompatibility is the way these two modules represent a datacenter. CloudSim defines class DataCenter, which is extended for purpose of intra-datacenter topology simulation by class NetworkDataCenter and for MapReduce simulation by class CloudDataCenter. This forces the user to instantiate one or the other but makes utilization of both features simultaneously impossible.

As mentioned above, CloudSim is mostly oriented at simulating multi-datacenter setups. That shows on the way it calculates data transfer time between two datacenters. This functionality is provided by class NetworkTopology. Topology is initialized from a *BRITE* file and then a matrix of node-to-node throughput is derived from it. Unfortunately, values in this matrix never change and thus make the simulator ignorant to the influence several flows sharing a link have on each other.

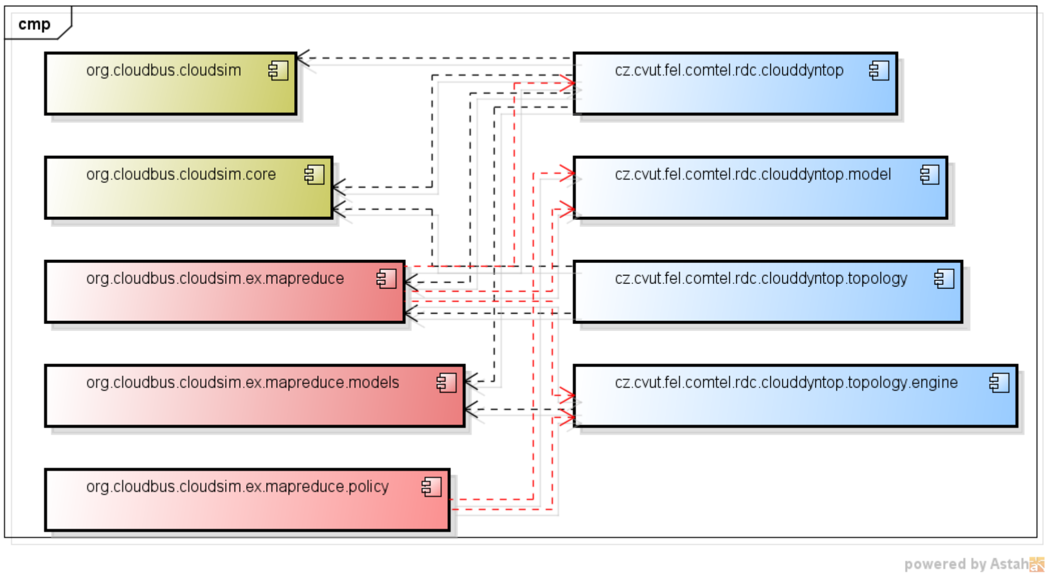
cmp
org.cloudbus.cloudsim
org.cloudbus.cloudsim.core
org.cloudbus.cloudsim.ex.mapreduce
org.cloudbus.cloudsim.ex.mapreduce.models
org.cloudbus.cloudsim.ex.mapreduce.policy
cz.cvut.fel.comtel.rdc.clouddyntop
cz.cvut.fel.comtel.rdc.clouddyntop.model
cz.cvut.fel.comtel.rdc.clouddyntop.topology
cz.cvut.fel.comtel.rdc.clouddyntop.topology.engine
powered by Astah

Figure 4.1: CloudDynTop component diagram - illustration of dependencies between CloudSim (yellow components), CloudSimEx MapReduce (red components) and our new CloudDynTop (blue components). Dependencies of CloudDynTop on external components are illustrated by black dotted lines, dependencies of CloudSimEx MapReduce on CloudDynTop are illustrated by red dotted lines. CloudSim components are not dependent on CloudDynTop.

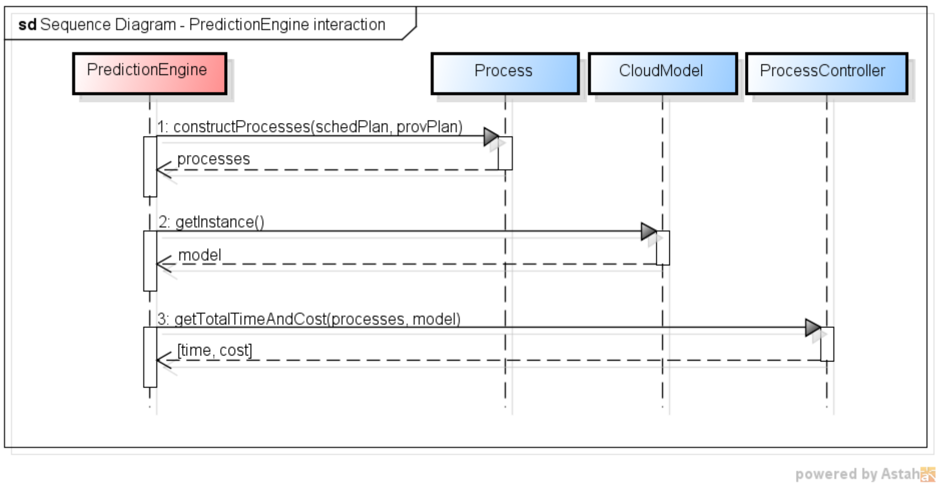
sd Sequence Diagram - PredictionEngine interaction
PredictionEngine
Process
CloudModel
ProcessController
1: constructProcesses(schedPlan, provPlan)
processes
2: getInstance()
model
3: getTotalTimeAndCost(processes, model)
[time, cost]
powered by Astah

Figure 4.2: Sequence Diagram - Interactions between PredictionEngine and CloudDynTop

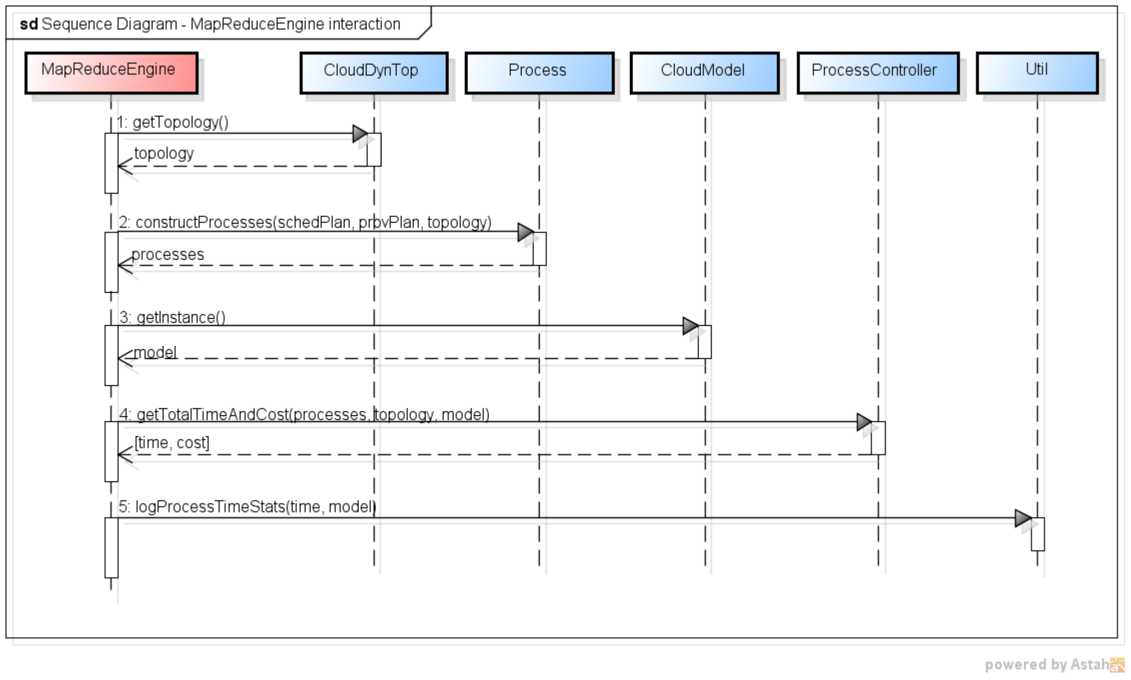
sd Sequence Diagram - MapReduceEngine interaction
MapReduceEngine
CloudDynTop
Process
CloudModel
ProcessController
Util
1: getTopology()
topology
2: constructProcesses(schedPlan, prbvPlan, topology)
processes
3: getInstance()
model
4: getTotalTimeAndCost(processes, topology, model)
[time, cost]
5: logProcessTimeStats(time, model)
powered by Astah

Figure 4.3: Sequence Diagram - Interactions between MapReduceEngine and CloudDynTop

Since our simulations greatly depend on the mutual influence of flows and on the ability to simulate different types of topology, it became necessary to extend CloudSimEx significantly. We implemented a new module - CloudDynTop - that provides the functionality of computing a MapReduce job's running time and topology representation to CloudSimEx MapReduce. Mutual dependencies between CloudSim, CloudSimEx MapReduce and CloudDynTop are illustrated on component diagram on Figure 4.1.

Details of dependencies of CloudSimEx MapReduce on CloudDynTop - the actual interactions are illustrated on sequence diagrams on Figure 4.3 and Figure 4.2. CloudDynTop's functionality of computing job's finish time is called from PredictionEngine and MapReduceEngine classes of CloudSimEx MapReduce.

4.2 MapReduce Job Finish Time Calculation

The CloudDynTop module keeps an instance of the simulated topology and takes input from CloudSimEx MapReduce. The provided input is in the form of scheduling (mapping between tasks and VMs) and provisioning plan (mapping between VMs and network nodes). Based on this input, CloudDynTop calculates the time needed for the tasks included in the scheduling plan to finish. For this purpose, approach of simulation with flexible time step has been used.

The scheduled tasks are used to construct instances of classes Flow and Computation. The relationship of these classes is illustrated on Figure 4.4. The Flow class represents the process of transmission of data between two nodes, the Computation class represents the processing of the data on a node. Each *Flow* and *Computation* has the information

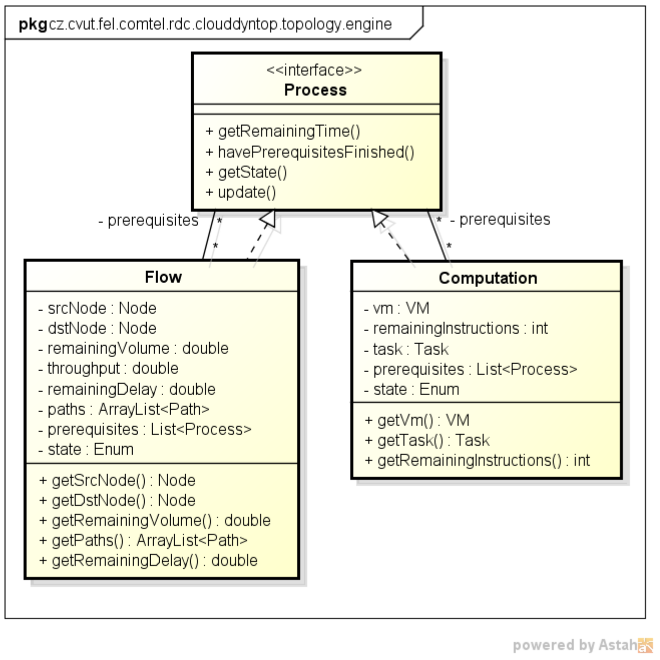
pkg cz.cvut.fel.comtel.rdc.clouddyntop.topology.engine
<<interface>>
Process
+ getRemainingTime()
+ havePrerequisitesFinished()
+ getState()
+ update()
- prerequisites
*
*
* - prerequisites
*
Flow
- srcNode : Node
- dstNode : Node
- remainingVolume : double
- throughput : double
- remainingDelay : double
- paths : ArrayList<Path>
- prerequisites : List<Process>
- state : Enum
+ getSrcNode() : Node
+ getDstNode() : Node
+ getRemainingVolume() : double
+ getPaths() : ArrayList<Path>
+ getRemainingDelay() : double
Computation
- vm : VM
- remainingInstructions : int
- task : Task
- prerequisites : List<Process>
- state : Enum
+ getVm() : VM
+ getTask() : Task
+ getRemainingInstructions() : int
powered by Astah

Figure 4.4: Class diagram capturing relationship of classes Flow and Computation. These classes implement the Process interface. They are used to represent processes a MapReduce job composes of - flows of data from the data source to Mapper nodes and from Mappers to Reducer nodes over the network and the computations on Mapper and Reducer nodes.

about how long will it take to finish. In case of Computation this is derived form number of remaining instructions and MIPS of the node where the VM running this task is provisioned. In case of Flow this is derived from the number of MB yet to be transmitted and bottleneck throughput and delay of the path taken through the network. Bottleneck throughput of a path is determined by Algorithm 3.

The processes composing up a MapReduce job generally cannot run all in parallel. Map computations need to wait for transmission of data from the data source, transfer of intermediary data from a Mapper to Reducers can start only after the corresponding Map computation has finished and similarly Reduce computations have to wait for intermediary data from Mappers. To implement these serialization barriers, each Process has a list of prerequisite processes and can only enter running state after all its prerequisites have finished.

For every Map task, the following processes are created:

- a Flow f from data source node to the task's node with no prerequisites,
- a Computation c on the task's node with the Flow f as a prerequisite and
- for each Reduce task one Flow from the Mapper's node to the Reducer's node the Computation c as prerequisite.

For each Reduce task a Computation on the task's node is created with the inbound Flows from Mapper nodes as prerequisites.

The running time of such set of processes is computed using the Algorithm 1.

Algorithm 1 Calculation of finish time of a set of processes

```
Require: $proc$: collection of Flows and Calculations
  $queued \leftarrow proc$
  $running \leftarrow$ empty list
  while $queued$ ¬empty or $running$ ¬empty do
      for $p$ in $queued$ do
          if $p$.allPrerequisitesFinished then
              $p.state \leftarrow RUNNING$
              $running$.add($p$)
              $queued$.remove($p$)
              if $p$ is Flow then
                  for $link$ in $p.path$ do
                      $link.passingPathsCnt++$
                  end for
              end if
          end if
      end for
      sort $running$ by remaining time (ascending)
      $rt \leftarrow$ smallest remaining time among $running$
      for $p$ in $running$ do
          $p$.updateProgress($rt$)
          if $p$.hasFinished then
              $running$.remove($p$)
              if $p$ is Flow then
                  for $link$ in $p.path$ do
                      $link.passingPathsCnt--$
                  end for
              end if
          end if
      end for
  end while
```

4.3 Topology Representation

CloudDynTop module network topology representation is implemented in package cz.cvut.fel.comtel.rdc.clouddyntop.topology.engine. Class diagram of topology representation is provided on Figure 4.5.

A network topology is represented by an instance of class GraphController in form of collection of Node and Link class instances (see attributes of GraphController class nodes and links respectively). GraphController class also acts as a Controller for the topology representation functionality as it implements the routing Algorithm 2 (method calculate FloydWarshall()) and throughput allocation algorithm described in subsection 4.3.2 (method

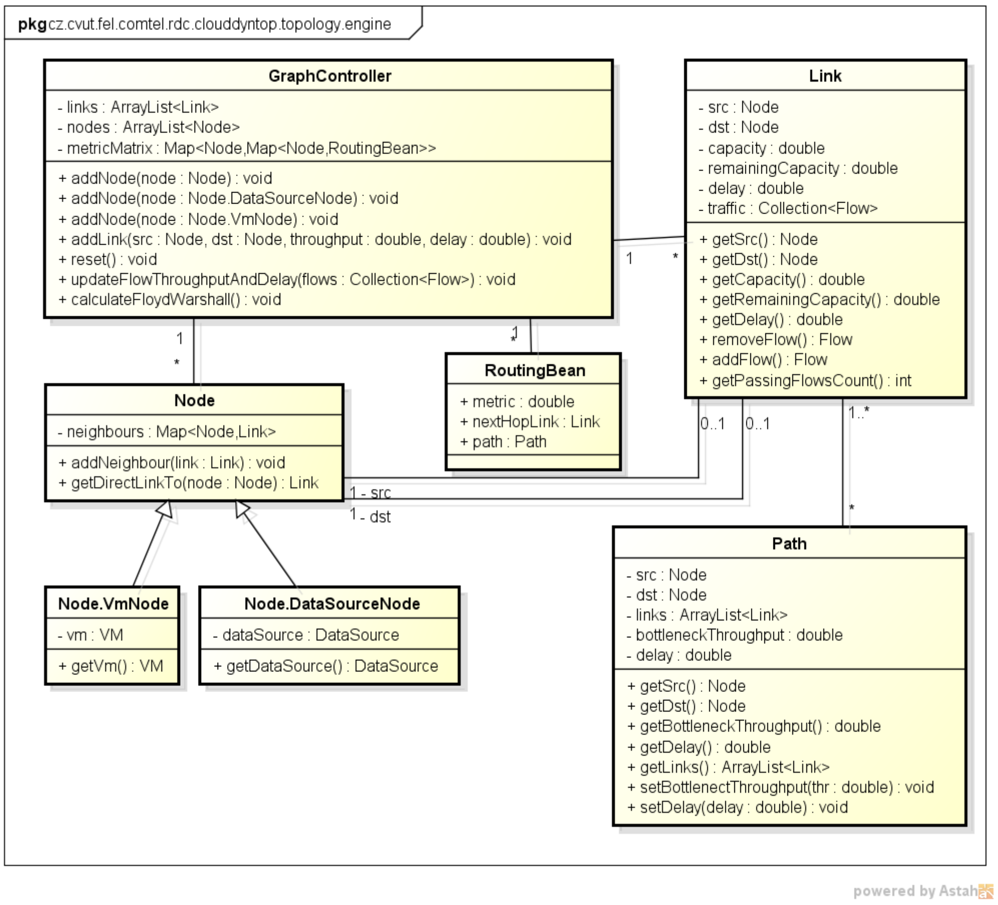
pkg cz.cvut.fel.comtel.rdc.clouddyntop.topology.engine
GraphController
- links : ArrayList<Link>
- nodes : ArrayList<Node>
- metricMatrix : Map<Node,Map<Node,RoutingBean>>
+ addNode(node : Node) : void
+ addNode(node : Node.DataSourceNode) : void
+ addNode(node : Node.VmNode) : void
+ addLink(src : Node, dst : Node, throughput : double, delay : double) : void
+ reset() : void
+ updateFlowThroughputAndDelay(flows : Collection<Flow>) : void
+ calculateFloydWarshall() : void
Link
- src : Node
- dst : Node
- capacity : double
- remainingCapacity : double
- delay : double
- traffic : Collection<Flow>
+ getSrc() : Node
+ getDst() : Node
+ getCapacity() : double
+ getRemainingCapacity() : double
+ getDelay() : double
+ removeFlow() : Flow
+ addFlow() : Flow
+ getPassingFlowsCount() : int
RoutingBean
+ metric : double
+ nextHopLink : Link
+ path : Path
Node
- neighbours : Map<Node,Link>
+ addNeighbour(link : Link) : void
+ getDirectLinkTo(node : Node) : Link
Node.VmNode
- vm : VM
+ getVm() : VM
Node.DataSourceNode
- dataSource : DataSource
+ getDataSource() : DataSource
Path
- src : Node
- dst : Node
- links : ArrayList<Link>
- bottleneckThroughput : double
- delay : double
+ getSrc() : Node
+ getDst() : Node
+ getBottleneckThroughput() : double
+ getDelay() : double
+ getLinks() : ArrayList<Link>
+ setBottlenectThroughput(thr : double) : void
+ setDelay(delay : double) : void
1
*
1
*
1
*
0..1
0..1
1..*
*
1 - src
1 - dst
powered by Astah

Figure 4.5: Class diagram of topology representation

updateFlowThroughputAndDelay(Collection<Flow>)). The calculateFloydWarshall() method only needs to be called once after the topology has been initialized, as it stores all the routing information in the metricMatrix attribute. Methods addNode(*) and addLink(Node, Node, double, double) serve as interface for initialization of a specific topology.

Each instance of Link class has references to two instances of Node class. Although these references are stored in attributes called src and dst, their order is irrelevant since the link represented by this class is bidirectional. Other important attributes of a Link are its capacity and delay. For purpose of consistent throughput distribution among flows passing through a link described in subsection 4.3.2, Link class also keeps information about throughput that hasn't been allocated to any flow yet (attribute remainingCapacity), as well as information about number of flows traversing it (attribute traffic - since a link also needs to keep track of how much throughput is which flow using and thus how much throughput is freed up when a flow finishes, the attribute traffic references instances of Flow class instead of keeping a simple counter of passing flows).

Both Node instances referenced by src and dst attributes of a single Link instance also keep a reference to each other. Nodes store these cross-node references in a HashMap (see attribute neighbours of class Node), where the adjacent node is the key and the link connecting these nodes is the value. The Node class is extended by two classes - Node.VmNode representing the machine hosting a VM, and Node.DatasourceNode representing the data source. This distinction is necessary so that CloudDynTop module would be able to associate topology nodes with VM resp. DataSource objects of CloudSimEx MapReduce.

4.3.1 Routing

The routing algorithm serving to determine which links will travers a flow between two nodes implemented in CloudDynTop is the Floyd-Warshall algorithm [13] illustrated on Figure 2.

This algorithm only has to be evaluated once after all nodes and links have been added to the GraphController instance. The results of the algorithm are stored in metricMatrix

Algorithm 2 Floyd-Warshall routing algorithm

```
Require: $nodes$: collection of nodes in the topology, $links$: collection of links in the topology
  $metricMatrix \leftarrow \{\}$
  for $n$ in $nodes$ do
    $metricMatrix.put(n, \{\})$
    for $nn$ in $nodes$ do
      if $n = nn$ then
        $metricMatrix[n][nn].metric \leftarrow 0$
      else
        $metricMatrix[n][nn].metric \leftarrow \infty$
      end if
      $metricMatrix[n][nn].link \leftarrow null$
    end for
  end for
  for $link$ in $links$ do
    $metricMatrix[link.src][link.dst].metric \leftarrow \textsc{metric}(link.capacity, link.delay)$
    $metricMatrix[link.src][link.dst].link \leftarrow link$
  end for
  for $n$ in $nodes$ do
    for $nn$ in $nodes$ do
      if $\neg(n = nn)$ then
        for $nnn$ in $nodes$ do
          if $\neg(nnn = nn \text{ or } nnn = n)$ then
            $bypass \leftarrow metricMatrix[nn][n].metric + metricMatrix[n][nnn].metric$
            if $bypass < metricMatrix[nn][nnn].metric$ then
              $metricMatrix[nn][n].metric \leftarrow bypass$
              $metricMatrix[nn][n].link \leftarrow metricMatrix[nn][n].link$
            end if
          end if
        end for
      end if
    end for
  end for
```

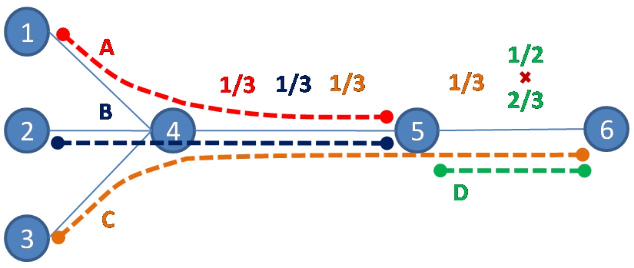

Figure 4.6: Available share versus fair share throughput allocation. All the links have same capacity. Throughput of flow C is determined by the link between nodes 4 and 5, therefore flow C can only use 1/3 of capacity of the link between nodes 5 and 6. Available share allocation will assign the remaining 2/3 to flow D, whereas fair share allocation would only assign 1/2 of the link's capacity to flow D.

attribute of GraphController in form of two levels of nested HashMap. The outer HashMap's entry key is the source node of a hypothetical flow, the entry key of the inner HashMap is the destination node. Value of the inner HashMap's entries is an instance of RoutingBean class containing the metric value for that path, next hop link and an instance of Path class representing the path taken by that hypothetical flow between the source and destination node.

The value of metric for a link between two nodes is computed using the following formula:

$$m = 256 \cdot (\frac{1250}{throughput} + 100000 \cdot delay) \tag{4.1}$$

where throughput is in MB and delay is in seconds. This metric follows the formula form metric of EIGRP routing protocol described in [24]. The advantage of this type of metric is that it takes into account both throughput and delay of a link.

4.3.2 Throughput Allocation

Allocation of links' throughput to flows passing through them has been calculated using Algorithm 3. Determining throughput of a flow just as the fair share of the link with the lowest capacity among links the flow is passing through would lead to suboptimal utilization of some of the links in the network. Algorithm 3 allows for assigning higher throughput than a fair share of a link's capacity to a flow if possible. We call this approach 'available share allocation'. The difference between the fair share and available share allocation is illustrated on Figure 4.6.

4.4 Selection of Optimal Scheduling Plan

To select the optimal scheduling plan, the CloudSimEx MapReduce module provides Branch-and-Bound (BB) algorithm implemented in class DecisionTree. This implementation has only tested running time of the candidate scheduling plan against a predefined threshold specified by user in the configuration of the simulator prior to launching the simulation.

Algorithm 3 Available share allocation

```
Require: $links$: collection of links in the topology, $flows$: flows passing through the topology
  while $links$ ¬empty do
      $bl \leftarrow minAvailableShare(links)$
      $as \leftarrow bl.availableCapacity/bl.passingPathsCnt$
      for $flow$ in $flows$ do
          $flow$.setBottleneckThroughput($as$)
          for $link$ in $flow.path$ do
              $link.availableCapacity \leftarrow link.availableCapacity - as$
              $link.passingPathsCnt - -$
          end for
      end for
      $links$.remove($bl$)
  end while
```

We have improved this algorithm so that running time is evaluated in each node of the scheduling space tree, e.g. for each candidate (even though incomplete) scheduling plan and the algorithm only proceeds with testing that node's child branches if the running time of that node is shorter than the shortest time of so far evaluated leafs.

To increase the performance of the algorithm, we have introduced two constraints to the scheduling problem:

1. The first one is elimination of permutations in Map task placement. Since Mappers receive the same volume of input data, permuting the Mappers over the selected set of VMs makes no difference with respect to the duration of flows from the data source to the Map nodes. In contrast, all permutations in Reducer placement are evaluated, and since the topologies simulated in this thesis are highly symmetrical, all possible relative positions of Mappers and Reducers are explored. This constraint greatly increases performance of the scheduler, because in MapReduce scenarios the number of Reducers is almost always smaller than the number of Mappers.

2. The second constraint allows only one task to be scheduled on a node. This presents another measure of trimming down the size of the scheduling space tree and since this thesis studies only influence of network topology on performance of MapReduce, it is an acceptable constraint.

The final version of the scheduling algorithm is described in Algorithm 4.

The DecisionTree class implements the Runnable interface and takes as argument in its constructor method the initial VM. This way, the scheduling space can be traversed concurrently by launching several instances of the DecisionTree class in multiple threads with different starting nodes.

Algorithm 4 Branch-and-Bound scheduling algorithm

```
Require: $VMs$: collection of all available VMs, $tasks$: collection of this job's tasks,
  $topology$: representation of simulated topology
  $minTime \leftarrow Inf$
  $solution \leftarrow []$
  procedure SEARCH($currTreeNode$, $candidateVMs$, $idx$)
      for $i$ in 0 to $candidateVMs.length - 1$ do
          $node \leftarrow currTreeNode + candidateVMs[i]$
          $time \leftarrow$ RUNNINGTIME($node$, $tasks$[0 to $node.length - 1$], $topology$)
          if $time < minTime$ then
              if ISLEAF($node$) then
                  $minTime \leftarrow time$
                  $solution \leftarrow node$
              else
                  if ISNEXTTASKREDUCER($node$) then
                      $nextIdx \leftarrow 0$
                  else
                      $nextIdx \leftarrow idx + i + 1$
                  end if
                  $nextCandidates \leftarrow []$
                  for $j$ in $nextIdx$ to $VMS.length - 1$ do
                      if $\neg nextCandidates.contains(VMs[j])$ then
                          $nextCandidates.add(VMS[j])$
                      end if
                  end forSEARCH($node$, $nextCandidates$, $nextIdx$)
              end if
          end if
      end for
  end procedure
  procedure ISLEAF($node$) return $node.length == tasks.length$
  end procedure
  procedure ISNEXTTASKREDUCER($node$) return $node.length < tasks.length$ and
  $tasks[node.length]$ is $Reducer$
  end procedure
```

Chapter 5

CloudDynTop simulator deployment

This chapter provides description of input and output files and arguments needed to run CloudDynTop simulations, as well as utility scripts for generating the input files and configuring and launching CloudDynTop.

5.1 CloudDynTop input arguments and files

CloudDynTop simulator is configured through command line arguments. The arguments are passed in the following format: key_word <space> value. The keywords, accepted values and description of the arguments are provided in Table 5.1.

The experiment definition file is a serialization of an Experiment class instance in YAML format required by CloudSimEx MapReduce. Among other experiment parameters such as user class policies, job submission time or job execution time and cost budget it defines path to the job definition file. Since the only part of the experiment definition file that was relevant to the simulations presented in this thesis was the path to the job definition file, the experiment definition file won't be described in more detail.

The job description file is also a configuration file of CloudSimEx MapReduce in YAML format. It contains serialization of an instance of the Job class. An example of such a file is presented on Figure 5.1.

Values in the mapTasks field are:

- the number of tasks of that description,
- input data volume for one such task in MB,
- millions of instructions one such task will process,
- a mapping describing distribution of intermediary data from each Mapper among Reducers (also in MB).

Table 5.1: Command line arguments of CloudDynTop

Key word	**Value**	**Description**
node.count	integer	Number of servers in topology
mapper.count	integer	Number of Mappers
reducer.count	integer	Number of Reducers
experiment	string	Path to experiment definition file
matlab.function.name	string	Name prefix of the generated Matlab function.
topology.type	see Table 5.2	Type of topology. Available topologies are listed in Table 5.2
wastefull.routing	true/false	Determines whether fair-share throughput allocation (string "true") or available-share throughput allocation (string "false") should be used.
multiple.tasks.per.vm	true/false	Determines whether the scheduler should consider scheduling plans where multiple tasks are assigned to the same machine.
branch.elimination	true/false	Determines whether the Branch-and-Bound scheduler should eliminate branches before reaching leafs if performance of their node is worse than performance of a previous leaf.
topology.bcube.n	integer	Optional argument. Parameter N of BCube topology
topology.bcube.k	integer	Optional argument. Parameter K of BCube topology
policy.bb.force2exit	integer	Optional argument. Time after which the Branch and Bound scheduler is forced to stop. If not specified, the scheduler evaluates all feasible scheduling plan candidates.
mind.deadlines	true/false	Optional argument. Determines whether the policy.bb.force2exit argument should be ignored (string "false") or not (string "true").
concurrent.threads	integer	Optional argument. Number of threads of the Branch and Bound scheduler that are allowed to run simultaneously.

```
!!org.cloudbus.cloudsim.ex.mapreduce.models.request.Job
dataSourceName: DS_0
mapTasks:
        - [10, 640, 640,
                {
                        Reducer_0: 19904,
                        Reducer_1: 6143,
                        Reducer_2: 1205
                }
        ]
reduceTasks:
        - [Reducer_0, 19904]
        - [Reducer_1, 6143]
        - [Reducer_2, 1205]
```

Figure 5.1: Example of job description specified in YAML format

Values in the reduceTasks field are:

- reducer name,
- millions of instructions the task will process.

To make configuration of job parameters easier a Python script for creation of the experiment and job description files automatically has been implemented, as described in following subsection.

CloudSimEx MapReduce also requires files simulation.properties and custom_log.properties, but since these files are not used for purposes of the presented simulations, we refer the reader to the documentation of CloudSimEx MapReduce [2] for more details about these files.

5.1.1 Utility scripts

The following Python scripts have been implemented in order to make configuration of CloudDynTop and running it more flexible:

Table 5.2: Topology types supported by CloudDynTop

Enum Value	**Topology**
HIERARCHICAL	3-tier hierarchical topology as described in subsection 6.1.1
BCube	BCube as described in subsection 6.1.2
BCube_T	BCube as described in subsection 6.1.4
CamCube	CamCube as described in subsection 6.1.5
MapReduce	MapReduce topology as described in subsection 6.1.3

generateSimulation.py: This script is used to generate the experiment and job description files. It takes two command line arguments: i) path to working directory of CloudDynTop and ii) number of realizations of the exponential and normal distribution of intermediary data to generate.

generateInterDataDistribution.py: This script is not directly used by the user, but it is called by the generateSimulation.py script. It implements the functionality of generating exponential and normal distribution of intermediary data used in the simulations presented in this thesis.

run.py: This script sets the values of CloudDynTop's arguments and launches the CloudDynTop.jar, passing it these parameters. It takes three command line arguments: i) label of the simulation scenario to be launched (see description below), ii) number of realizations of this simulation to be run and iii) the index of the initial realization of this simulation to be run

CloudDynTopSim.py: It defines values of CloudDynTop's arguments for individual simulation scenarios. This script is referenced from run.py script, rather than being directly called by the user.

The labels of the simulation scenarios follow the following grammar:

```
{BC | BC1 | CC | HI | MR}_{M10_R03 | M15_R04 | M20_R05}_{E500}_{EXP | NORM | UNI}
```

where:

- BC specifies the BCube topology, BC1 specifies the $BCube_1$ topology, CC specifies the CamCube topology, HI specifies the hierarchical topology and MR specifies the MapReduce topology
- M10_R03 specifies a job with 10 Mappers and 3 Reducers, M15_R04 specifies a job with 15 Mappers and 4 Reducers and M20_R05 specifies a job with 20 Mappers and 5 Reducers
- EXP specifies exponential data distribution, NORM specifies normal data distribution and UNI specifies uniform data distribution

5.2 CloudDynTop output files

CloudDynTop produces its output in form of Matlab function files. Each run of the simulator produces two functions:

1. Function providing information about start and finish times of flows and computations of the simulated MapReduce job and utilization of the links in the topology.

   ```
   <name>_<yyyy><MM><dd><hh><mm><ss><ms>
   ```

2. Function generating description of the simulated topology

   ```
   <name>_<yyyy><MM><dd><hh><mm><ss><ms>_top
   ```

where:

- *name* is the value of CloudDynTop's argument matlab.function.name,
- yyyy is the year,
- MM month (2 digits),
- dd day of month,
- hh hour in 24-hour format,
- mm minutes,
- ss seconds and
- ms milliseconds of the time the simulator has been launched.

The first listed function returns 4 variables:

1. flows - Matrix with one row for each data transfer flow triggered by the simulated job. Columns of this matrix are: i) source VM id, ii) destination VM id, iii) volume of transmitted data in MB, iv) start time of the flow, v) finish time of the flow, vi) id of the source topology node, vii) id of the destination topology node.

2. computations - Matrix where each row represents one computation triggered by the job. Columns of this matrix are: i) task type (0 represents Mapper, 1 represents Reducer), ii) task's id, iii) VM id, iv) computation's start time, v) computation's finish time.

3. linkUtilization - Matrix where row index corresponds to index of a link's source node and column index corresponds with the link's destination node. Values in the matrix describe the number of flows that have passed through that link.

4. nodeIDs - A vector providing mapping of node indexes used in linkUtilization matrix (index of this vector's element) to topology node ids (values of this vector's elements).

An example of a portion of this function is given on Figure 5.2.

The second listed function provides description of the simulated topology in form of 2 variables:

1. id - IDs of the instances of Node class in the topology. The order they appear in this variable corresponds to the order of columns and rows in the adj matrix.

2. adj - matrix where each row resp. column corresponds to a node and value at position i, j describes, whether nodes i and j are directly connected.

```
function [flows, comp, linkUtil, nodeIDs] = getStats_20150430124744294661()
%HIERARCHICAL n19 m15 r4 fs:false18000000
        flows = [
                -2, 14, 640.0, 0.0, 9.660599999999999, 175, 189;...
                14, 4, 18379.0, 16.0554, 291.4034076190477, 189, 179;...
                -- text ommitted --
        ];
        comp = [
                0, 34, 33, 9.660599999999999, 16.0554;...
                0, 35, 34, 9.660599999999999, 16.0554;...
                -- text ommitted --
        ];
        linkUtil = [
                0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;...
                -- text ommitted --
        ];
        nodeIDs = [
        175, 174, 201, 198, 189, 202, 200, 195, 179, 196, ...
        183, 197, 187, 191, 190, 186, 188, 199, 194, 176, 192, 193, ...
        177, 180, 178, 182, 181, 185, 184];
```

Figure 5.2: Example of output of CloudDynTop providing process time statistics.

Determining a job's finish time is simply the matter of finding the maximum finish time among the computations. This could be done using the following Matlab code:

```
function [finishTime, taskId] = getJobFinishTime(comp)
        % returns latest finish time and ID of corresponding task
        [finishTime, taskRowIdx] = max(comp(:,5));
        taskId = comp(taskRowIdx, 2);
end
```

Chapter 6

Simulation setup

This chapter discusses the simulation scenarios designed to study the influence of network topology on MapReduce performance. The section 6.1 provides description of the simulated topologies. Simulated workloads and the process of generating them is described in section 6.2.

6.1 Simulated Topologies

The simulated datacenters consisted of the same number of nodes as the sum of Mappers and Reducers plus a data source node. A constraint of only one task per host has been added since this work studies influence of network topology on MapReduce performance and this constraint significantly lowered computational costs of finding the optimal scheduling plan. Hence the specified constraint doesn't byass the simulations.

Simulated topologies were:

- Hierarchical,
- BCube,
- "MapReduce",
- $BCube_1$ (see description below),
- CamCube.

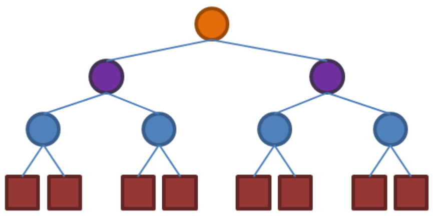

Figure 6.1: Illustration of the 3-Tier hierarchical topology

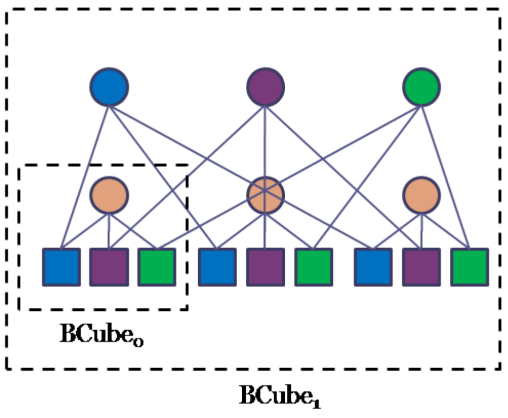

Figure 6.2: Illustration of BCube($N = 3$, $K = 1$). A level-k BCube(N, k) consists of N BCubes(N, $k-1$) and N^{k+1} switches. The ith switch in this BCube(N, k) connects to the ith host in each of its level-($k-1$) BCubes. In this figure, squares represent hosts and circles represent switches.

6.1.1 Hierarchical Topology

The Hierarchical topology is a tree topology consisting of 3 layers of switches - edge, aggregation and core. It has been configured so that four hosts connect to a single edge switch, 4 edge switches connect to a single aggregate switch and all aggregate switches connect to a single core switch. Illustration of this topology is given on Figure 6.1. All used links are the same with throughput of 1Gb/s.

6.1.2 BCube Topology

The BCube topology [14] has been introduced in paper [14]. It's a recursively defined topology that can be described by 2 parameters: N and K, where

- N determines the number of hosts in the level-0 BCube (the lowest level of recursion),
- K defines index of the highest recursion level.

Illustration of a BCube($N = 3$, $K = 1$) is on Figure 6.2. For the purpose of our simulations, 1Gb/s links have been chosen for all connections in this topology. Chosen values for the K and N parameter of BCube shows Table 6.1. Number of hosts in that table includes Map nodes, Reduce nodes and a data source node.

Table 6.1: Parameters of BCube topology

Hosts	N	K
14	4	2
20	5	2
26	6	2

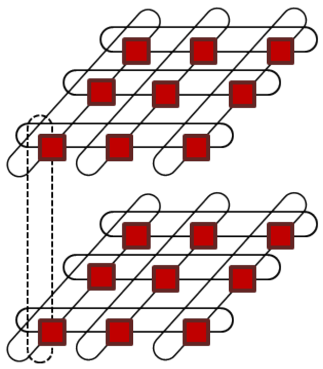

Figure 6.3: Illustration of 3D Torus topology used in CamCube network stack. Servers in the same plane are represented by rectangles of the same color. Each server connects to 6 of its neighbours, so that every row along each of the x, y and z axes forms a closed circle. The figure shows these connections only within each horizontal plane, but vertical planes are connected in the very same way.

6.1.3 MapReduce Topology

The MapReduce topology is included for reference. In follows data flow of MapReduce systems - each Map node is directly connected to the data source node as well as to each of the Reduce nodes, thus no link in the topology is traversed by more than one flow. This topology is the optimal topology for MapReduce, because each network flow has its own dedicated set of links and the only constraint is the physical capacity of those links. Unfortunately, this topology is not flexible and needs to be modified when the job dimensions change. An illustration is provided on Figure 1.2. All used links have also 1Gb/s of throughput.

6.1.4 $BCube_1$ Topology

The topology labeled as $BCube_1$ is a modified version of the BCube topology. The difference is that the as many nodes as there are Reduce tasks and the data source node are connected to the respective switches by links with 10 times higher throughput than the rest of the hosts (10Gb/s links).

6.1.5 CamCube Topology

The CamCube network stack [5] uses 3D Torus topology. In this topology, servers are connected directly to each other, each server connects to 6 of its neighbours. An illustration is provided on Figure 6.3.

6.2 Simulated Workloads

Simulated job dimensions were:

- 10 Mappers and 3 Reducers,
- 15 Mappers and 4 Reducers,
- 20 Mappers and 5 Reducers.

Each of these jobs has been simulated along with the following intermediary data distributions:

- exponential,
- normal,
- uniform.

Figure 6.4 shows one realization of these data distributions. The exponential and normal distribution have been simulated in 10 realizations.

Simulated type of MapReduce workload was expansion job (as formulated in [23]). A typical feature of this kind of workload is that the volume of intermediary data is greater than the volume of the job's input data. For our simulations we have chosen expansion ratio of 5, meaning that the sum of all intermediary data volumes produced by all Mappers totaled to 500% of the sum of input data volumes of all Mappers. This ratio has been chosen as a realistic value large enough to emphasize the influence of network traffic latency on MapReduce performance. Realizations of the exponential and normal distribution have been generated using the following procedure.

The distributions were sampled on interval $\langle 0;5\rangle$. The same number of pseudo-equidistant points as the number of reducers has been chosen on this interval, their exact value determined by this formula:

$$x_i = i \cdot \frac{R}{N} + |Norm(\mu = 0, \sigma^2 = 0.5)| \tag{6.1}$$

Here $i = 0, 1, ..., N-1$, R is the width of the sampling interval (here 5), N is the number of Reducers and $Norm(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2. The distributions have then been evaluated in these points, the exponential distribution using equation 6.2, and the normal distribution using equation 6.3.

$$E(x, A, \lambda) = A \cdot \lambda e^{-\lambda \cdot x} \tag{6.2}$$

$$N(x, A, \lambda) = A \cdot \frac{e^{\frac{x^2}{2 \cdot \sigma^2}}}{\sigma \cdot \sqrt{2 \cdot \pi}}, \tag{6.3}$$

where in both cases the choice of λ has been $\lambda = 0.7$ and A has been calculated so that the sum of the distribution's values in all sample points would meet the condition of intermediary data volume stated above.

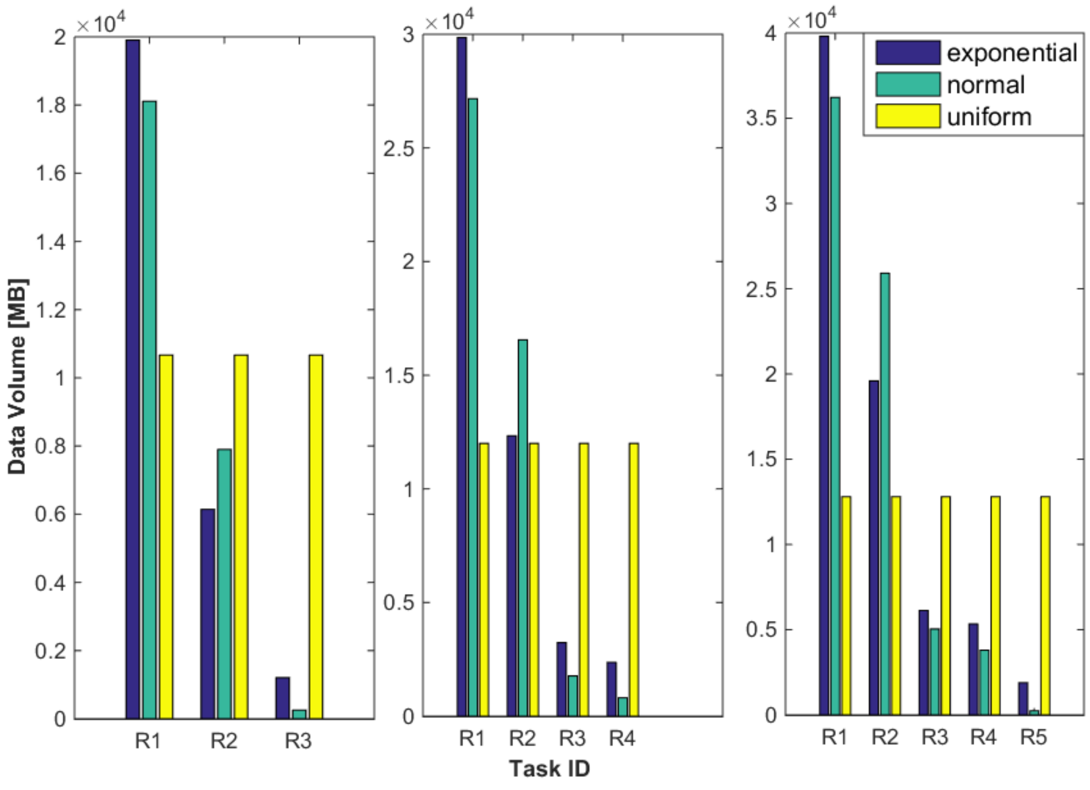

Figure 6.4: One realization of intermediary data distributions. The three charts correspond to different job sizes - jobs with 10 Mappers and 3 Reducers, 15 Mappers and 4 Reducers and 20 Mappers and 5 Reducers respectively. These charts illustrate the volume of data each Reducer (here represented by its ID R1 - R5) from every single Mapper. In each of the charts above each of the distribution functions is handling the same total data volume.

Chapter 7

Results

This chapter presents results of the conducted simulations. Section 7.1 discusses performance of individual MapReduce workloads on different topologies. The two proposed approaches to throughput allocation are also compared in that section. Performance of the simulator itself is evaluated in section 7.2.

7.1 MapReduce Performance

Main output of our simulations are finish times of the simulated workloads on individual topologies. These results are presented on Figure 7.1, Figure 7.2 and Figure 7.3.

In case of all distributions the best performance was achieved (as expected) by the MapReduce topology, where each flow has its own dedicated link. This topology represents optimal performance given the limited capacity of links in the network.

The CamCube topology performed only slightly worse than the MapReduce topology, especially in case of uniformly distributed intermediary data. When skew has been introduced into the distribution, CamCube has been outperformed by $BCube_1$, although that's not a fair comparison, since $BCube_1$ utilizes faster links to connect its Reduce nodes.

The BCube topology has performed very well, although not as well as CamCube. An interesting result is that increasing throughput of link connecting the Reducer nodes has had a great impact on the MapReduce job's performance. As mentioned above, the modified BCube topology - $BCube_1$ lead to the best performance (not considering the reference MapReduce topology) in cases where the intermediary data suffered from skew.

Performance on hierarchical topology has been the worst for all the simulated workloads, because of the bottlenecks between edge and aggregation and aggregation and core layer. Because of these bottlenecks the hierarchical topology has been clearly outperformed by the other simulated topologies.

Our simulations have also proven that the available share throughput allocation always leads to better performance than the fair share allocation. Comparison of performance of MapReduce job with 15 Mappers and 4 Reducers with exponential data distribution when using these two algorithms is shown on Figure 7.4.

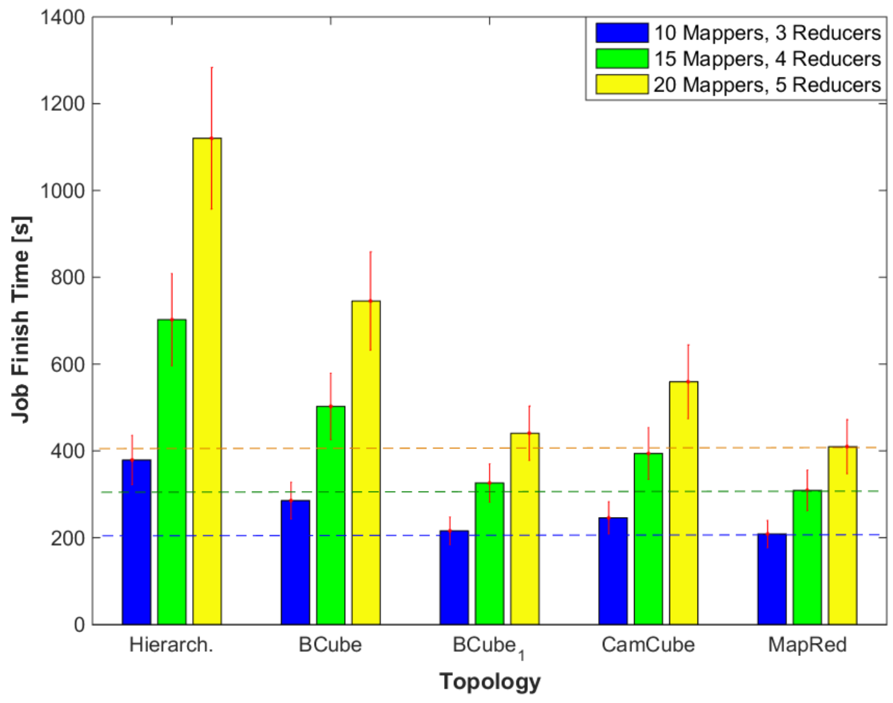

Figure 7.1: Finish time of jobs with exponential distribution of intermediary data.

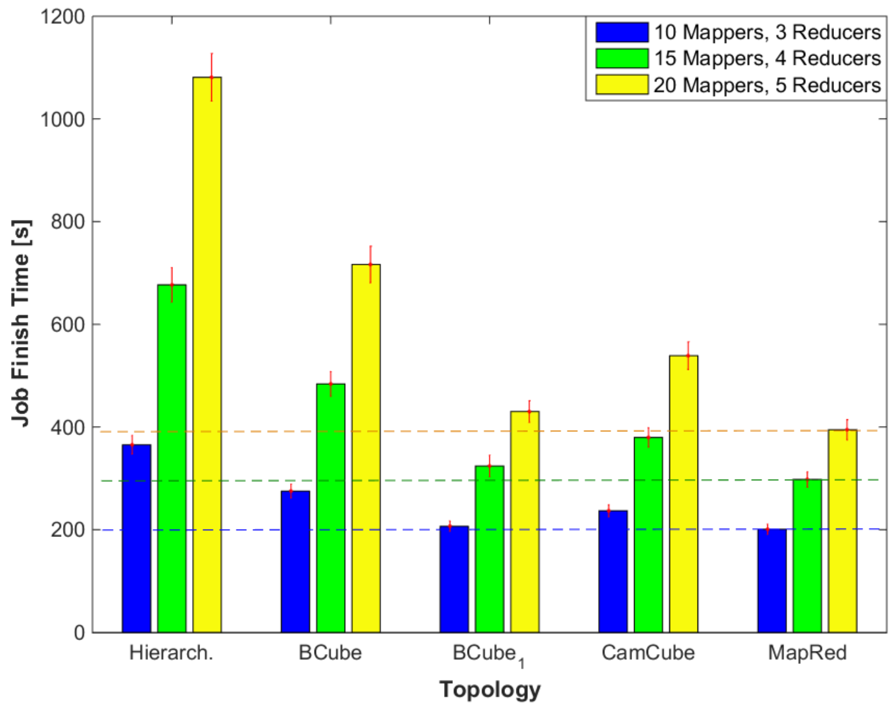

Figure 7.2: Finish time of jobs with normal distribution of intermediary data.

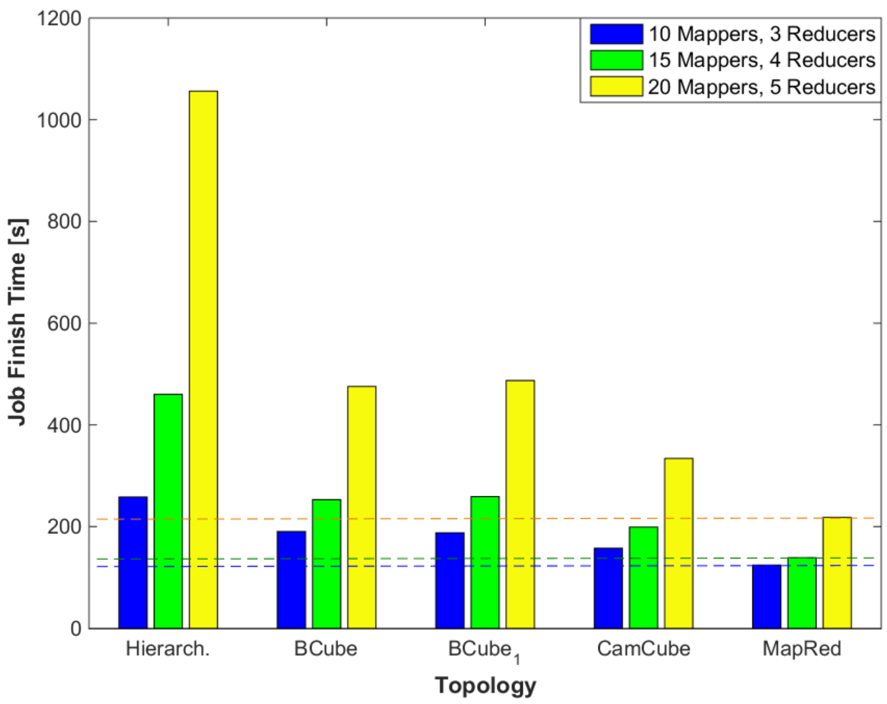

Figure 7.3: Finish time of jobs with uniform distribution of intermediary data.

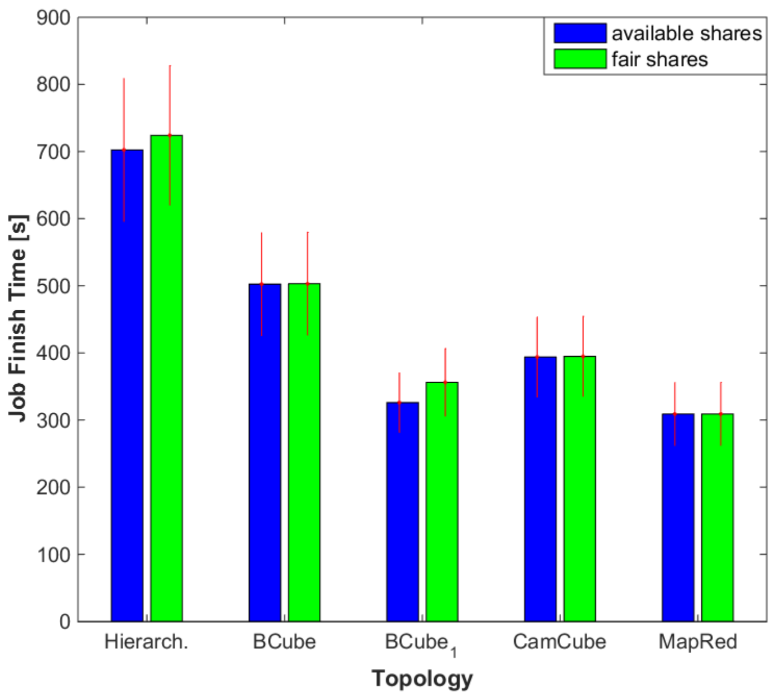

Figure 7.4: Comparison of job finish times when available- resp. fair-share throughput allocation has been used. In all cases available share allocation leads to better performance. The simulated job had 15 Mappers and 4 Reducers with exponential data distribution.

7.2 Simulator Performance

Another interesting result of our simulations is the performance of the simulator itself. Running times of the simulator are plotted on Figure 7.5. Since the most resource-heavy part of the simulation is the scheduler, these values show how well was the Branch-and-Bound algorithm able to eliminate branches of the scheduling space before reaching leafs, when looking for the optimal scheduling plan in case of different topologies.

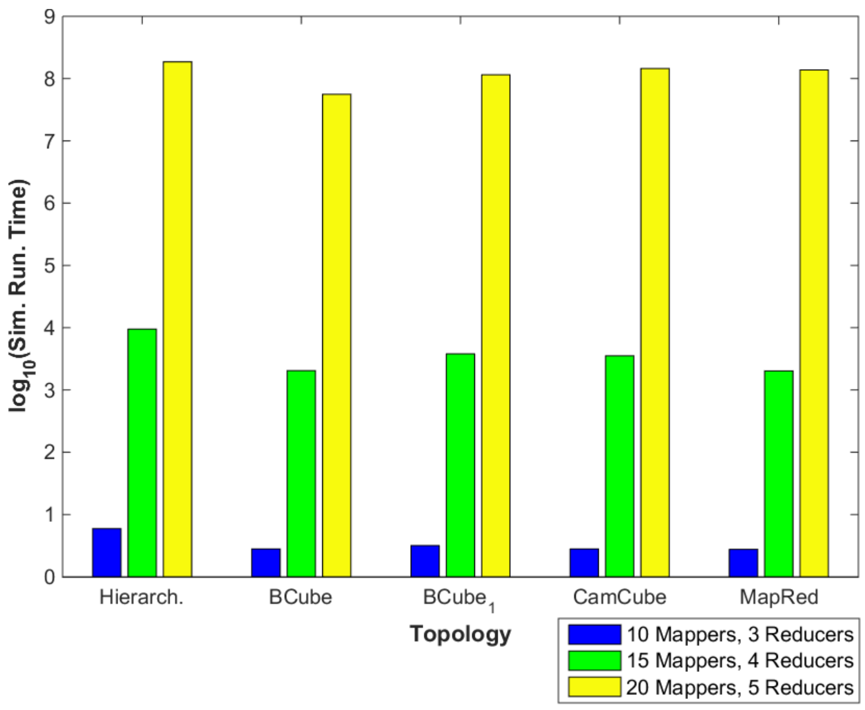

Figure 7.5: Mean of 10 realizations of simulator's running time in **logarithmic scale**. The values have been collected when simulation scenarios for shown topologies and job dimensions with exponential intermediary data distribution were being processed. The differences of running time for different topologies are caused by varying complexity of job finish time calculation on different topologies as well as by the Branch-and-Bound scheduling algorithm eliminating branches of the scheduling space tree at varying levels with varying frequencies (caused directly by the job finish time values).

Chapter 8

Conclusion

In this thesis we have presented setup and results of simulations studying performance of different types of MapReduce workload on different network topologies, as well as implementation of network topology simulation module - CloudDynTop - extending functionality of the CloudSimEx MapReduce simulator.

The Hierarchical topology has performed the worst in all of the simulation scenarios. Quite well has performed the BCube topology [14], although the CamCube topology [5] has performed even better. Increasing throughput of links connecting Reduce nodes to the rest of the BCube topology (topology $BCube_1$) had make it perform better than CamCube in cases where the distribution of intermediary data has been uneven, but at the cost of using faster links. In case of uniformly distributed intermediary data the CamCube topology performed even better than $BCube_1$. The MapReduce topology has met the expectation of having the best performance, but it is not a very practical topology as it is specifically designed to follow data flow of a specific MapReduce job.

Results of our simulations show that implementing a sophisticated topology such as BCube or CamCube in a datacenter can improve performance of MapReduce jobs in that cluster significantly. Best performance would be achieved by dynamically fitting the topology to the data flow of the workload being processed as suggests performance of the MapReduce topology in our simulations. This would be possible to achieve utilizing an SDN controller working together with a workload monitoring system. Design of such a system could be an object of future work.

The presented simulations mainly focused on the impact of data transfer over network. Computational overhead that is imposed on servers by switching and forwarding traffic, most significant in case of the BCube and CamCube topologies, has been neglected.

Presented results also show how the simulated topology affects running time of the simulator, due to different complexity of calculating the job finish time and varying effectivity of the Branch-and-Bound scheduling algorithm in pruning branches of the scheduling space tree.

The need to implement the CloudDynTop module became obvious after examining the source code of CloudSimEx MapReduce. As we mentioned earlier, this simulator doesn't account for network flow interactions in its topology simulation module. Also the Branch-and-Bound algorithm implemented in CloudSimEx's scheduler didn't meet our expectations, since it only eliminated branches that exceeded a predefined threshold as opposed to pruning branches that are certain to lead to a solution that is worse than a previously found one. The network topology simulation issue has been addressed by the implemented CloudDynTop module.

Future work on this project would include simulation of more topologies and workload types as well as running benchmark test on a real cluster. Results of these simulations and benchmarks should lead to derivation of a network topology management strategy for a SDN controller designed to optimize the topology for the running MapReduce workload.

Bibliography

[1] Apache hadoop. http://hadoop.apache.org/, cited on May 5th, 2015.

[2] Cloudsimex home page. https://github.com/Cloudslab/CloudSimEx/, cited on May 5th, 2015.

[3] Simjava home page. http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/, cited on May 5th, 2015.

[4] Yaml home page. http://yaml.org/, cited on May 5th, 2015.

[5] Hussam Abu-Libdeh, Paolo Costa, Antony Rowstron, Greg O'Shea, and Austin Donnelly. Symbiotic routing in future data centers. *ACM SIGCOMM Computer Communication Review*, 41(4):51–62, 2011.

[6] Mansoor Alicherry and TV Lakshman. Optimizing data access latencies in cloud systems by intelligent virtual machine placement. In *INFOCOM, 2013 Proceedings IEEE*, pages 647–655. IEEE, 2013.

[7] Kashif Bilal, Samee U Khan, Limin Zhang, Hongxiang Li, Khizar Hayat, Sajjad A Madani, Nasro Min-Allah, Lizhe Wang, Dan Chen, Majid Iqbal, et al. Quantitative comparisons of the state-of-the-art data center architectures. *Concurrency and Computation: Practice and Experience*, 25(12):1771–1783, 2013.

[8] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1175–1220, 2002.

[9] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[10] Henri Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 430–437. IEEE, 2001.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[12] Yuanquan Fan, Weiguo Wu, Haijun Cao, Huo Zhu, Wei Wei, and Pengfei Zheng. Lbvp: A load balance algorithm based on virtual partition in hadoop cluster. In *Cloud Computing Congress (APCloudCC), 2012 IEEE Asia Pacific*, pages 37–41. IEEE, 2012.

[13] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[14] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[15] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. *ACM SIGCOMM Computer Communication Review*, 38(4):75–86, 2008.

[16] Suhel Hammoud, Maozhen Li, Yang Liu, Nasullah Khalid Alham, and Zelong Liu. Mrsim: A discrete event based mapreduce simulator. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, volume 6, pages 2993–2997. IEEE, 2010.

[17] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: Saving energy in data center networks. In *NSDI*, volume 10, pages 249–264, 2010.

[18] Wagner Kolberg, Pedro De B Marcos, Julio CS Anjos, Alexandre KS Miyazaki, Claudio R Geyer, and Luciana B Arantes. Mrsg–a mapreduce simulator over simgrid. *Parallel Computing*, 39(4):233–244, 2013.

[19] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. A study of skew in mapreduce applications. *Open Cirrus Summit*, 2011.

[20] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 25–36. ACM, 2012.

[21] Charles E Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *Computers, IEEE Transactions on*, 100(10):892–901, 1985.

[22] Peng Qin, Bin Dai, Benxiong Huang, and Guan Xu. Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data. *arXiv preprint arXiv:1403.2800*, 2014.

[23] Kai Ren, Garth Gibson, YongChul Kwon, Magdalena Balazinska, and Bill Howe. Hadoop's adolescence; a comparative workloads analysis from three research clusters. In *SC Companion*, page 1452, 2012.

[24] Cisco Systems. Enhanced interior gateway routing protocol (eigrp) wide metrics. *Cisco White Paper*, 2012.

[25] Tomas Vondra and Jan Sedivy. Maximizing utilization in private IaaS clouds with heterogenous load. In *CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 169–173, 2012.

[26] Guanying Wang, Ali R Butt, Prashant Pandey, and Karan Gupta. Using realistic simulation for performance analysis of mapreduce setups. In *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*, pages 19–26. ACM, 2009.

[27] Guanying Wang, Ali Raza Butt, Prashant Pandey, and Karan Gupta. A simulation approach to evaluating design decisions in mapreduce setups. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*, pages 1–11. IEEE, 2009.

[28] Guohui Wang, TS Ng, and Anees Shaikh. Programming your network at run-time for big data applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 103–108. ACM, 2012.

[29] Weikuan Yu, Yandong Wang, and Xinyu Que. Design and evaluation of network-levitated merge for hadoop acceleration. *Parallel and Distributed Systems, IEEE Transactions on*, 25(3):602–611, 2014.

[30] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, pages 265–278. ACM, 2010.

Appendix A

Contents of attached CD

The attached CD contains the following files and folders:

dist/ Folder containing distribution of the CloudDynTop module integrated into the CloudSImEx MapReduce simulator, as well as scripts and configuration files used to run and evaluate the simulations presented in this thesis

dist/CloudDynTop_lib/ Folder containing the compiled libraries used by the CloudDynTop module

dist/inputs/ Folder containing YAML files defining simulated workloads

dist/matlab/ Folder containing the matlab scripts used to evaluate output of CloudDynTop, as well as files containing the results of the simulations presented in this thesis

dist/matlab/evalScript.m Matlab script used to plot the charts presented in this thesis

dist/matlab/getAvgFinishTimesMatrix.m Matlab function called by evalScript.m. It constructs a matrix of mean values and a matrix of std. variations of job finish times of individual simulation scenarios presented in this thesis

dist/matlab/getAvgJobFinishTime.m Matlab function called by getAvgFinishTimesMatrix.m to determine mean value and std. variation of finish times of one simulated workload

dist/matlab/getFinishTimesMatrix.m Matlab function called by evalScript.m. It constructs a matrix of job finish times of individual simulation scenarios presented in this thesis

dist/matlab/getJobFinishTime.m Matlab function called by getAvgFinishTimesMatrix.m to determine finish time of one simulated workload

dist/matlab/getStats_*.m Output files of CloudDynTop containing data presented in this thesis

dist/CloudDynTop.jar Executable JAR of CloudDynTop module integrated into CloudSimEx MapReduce simulator

dist/CloudDynTopSim.py Python module defining parameters of individual simulation scenarios

dist/custom_log.properties A configuration file of CloudSImEx MapReduce

dist/generateInterDataDistribution.py Python script called by generateSimulation.py to generated realizations of exponential and normal distributions of intermediary data

dist/generateSimulation.py Python script used to generate YAML files defining the simulation scenarios presented in this thesis. Usage of this script is described in subsection 5.1.1 of this thesis.

dist/run.py Python script used to configure and launch CloudDynTop. Usage of this script is described in subsection 5.1.1 of this thesis.

dist/sim_*.yaml Configuration files of CloudSimEx MapReduce defining individual simulation scenarios presented in this thesis

dist/simulation.properties A configuration file of CloudSimEx MapReduce

src/ Folder containing the source code of CloudDynTop and the source code of CloudSimEx MapReduce modified to integrate the CloudDynTop module

Kouba_Zdenek.pdf The PDF version of this thesis