Lecture Notes
Introduction to Cloud Computing

Theory and Practice

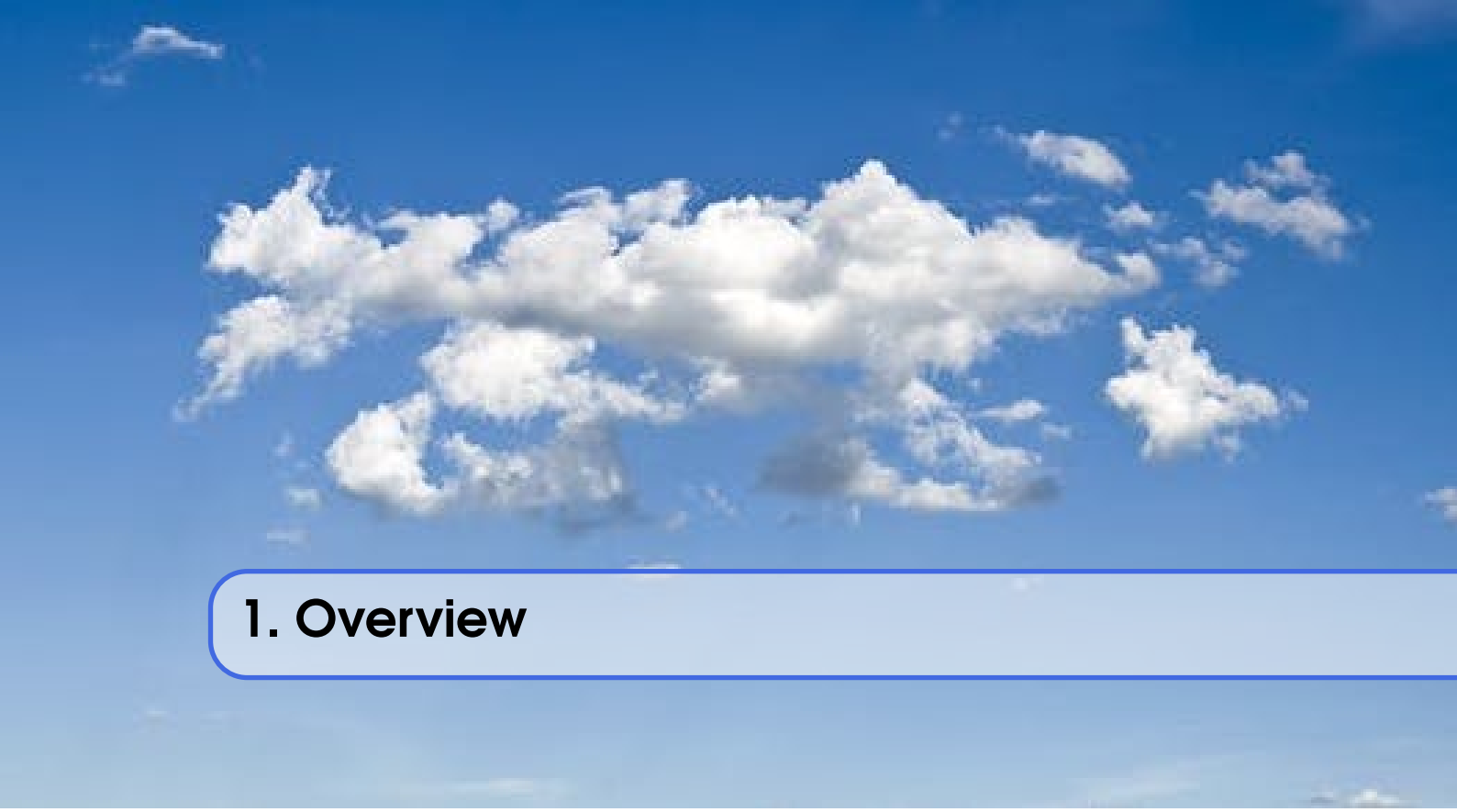
HTTPS://GITHUB.COM/CLOUDMESH/CLASSES

First printing, October 2017

Contents

F part-icloud.tex

Preface



1. Overview

F section/icloud/preface.tex

1.1 Cloud Computing

The course covers all aspects of the cloud architecture stack, from Software as a Service (large-scale biology and graphics applications), Platform as a Service (MapReduce (Hadoop), Iterative MapReduce (Twister) and NoSQL (HBase)), to Infrastructure as a Service (low-level virtualization technologies).

1.2 Class Summary

In this course you will learn basic concepts in Cloud Computing, including how to write your own software using key cloud programming models and tools to support data mining and data analysis applications.

1.3 Course Organization

This course is powered by Latex and ReStructuredText but is conducted in a more structured fashion with a mix of recorded lectures, programming labs and forum discussions. Each week we will post on Canvas the instructions as to work to be done. Note all grading and forum discussion will use Canvas and Piazza.

1.4 What Should I Know?

Cloud Computing online is a programming intensive course. It has similar requirements to the CS graduate level residential version. Students are expected to have weekly (or biweekly) programming homework. General programming experience with Windows or Linux using Java (2-3 years) and scripts is required. A background in parallel and cluster computing is a plus, although not necessary.

1.5 What Will I Learn?

At the end of this course, you will have learned key concepts in cloud computing and enough programming to be able to solve data analysis problems on your own.

1.6 Class Projects

The class has several projects that will allow students to get firsthand experience with the technologies taught here. Projects are performed on VirtualBox Appliances or academic clouds like FutureGrid.

1.7 Course Schedule

- Welcome Survey
- Unit 1 - Cloud Computing Fundamentals
- Unit 2 - How to Run VMs (IaaS)
- Unit 3 - How to Run MapReduce (PaaS)
- Unit 4 - How to Run Iterative MapReduce (PaaS)
- Mid-course assessment
- Unit 5 - How to Store Data (NoSQL)
- Unit 6 - Internet of Things
- Unit 7 - How to Build a Search Engine (SaaS)
- Post-course assessment
- Post-course Survey
- Test-course assessment

1.8 Class Progress Distribution

The course progress is the percentage of mandatory items completed for the course.

- Participation (10%)
- Exams (50%) - Midterm (20%), Final (30%)
- Written Assignments (10%)
- Projects (30%) - 8 Projects: [Hadoop Statistics (5), Hadoop PageRank (10), Hadoop Blast (10), HBase WordCount (5), Building an Inverted Index (10), Build a Search Engine (20), Harp PageRank (20), Harp Mini-batch K-Means (20)]

1.9 Lesson Order

Lessons in this course will be uploaded on a weekly basis. The difficulty of the topics covered in these online lectures is demonstrated by way of a color coding system. 'Green' is basic, 'Yellow' is intermediate, and 'Red' is advanced. We will build up to more difficult concepts as the class progresses.

1.10 Course Projects

Take part in online assignments that will teach you the course material in hands-on examples. You will apply actual code designed to calculate website page ranking, word count, and even build your own search engine. More than a dry tutorial, this is your chance to find out how some of the most widely used applications on the Internet work on a fundamental level.

1.11 Course Developer

Dr. Judy Qiu is an Associate Professor in the School of Informatics and Computing at Indiana University. Her research interests focus on data-intensive computing at the intersection of cloud and multicore technologies, with an emphasis on life science applications using MapReduce as well as traditional parallel and distributed computing approaches.

section/icloud/syllabus.tex

1.12 Schedule

No	Title	Level
Survey	Welcome Survey	
Unit 1	Cloud Computing Fundamentals	Basic
Unit 2	How to Run VMs (IaaS)	Intermediate
Unit 3	How to Run MapReduce (PaaS)	Advanced
Unit 4	How to Run Iterative MapReduce (PaaS)	Advanced
Exam	Mid-course assessment	Basic
Unit 5	How to Store Data (NoSQL)	Advanced
Unit 6	Internet of Things	Intermediate
Unit 7	How to Build a Search Engine(SaaS)	Advanced
Exam	Post-course assessment	Intermediate
Survey	Post-course Survey	
Survey	Test-course assessment	

1.13 Grade distribution

The course progress is the percentage of mandatory items completed for the course.

Task	Weight	Detail
Participation	(10%)	Lab attendance
Written Assignments	(10%)	Quizzes, homework
Projects	(30%)	8 Projects: [Hadoop Statistics (5), Hadoop PageRank (10), Hadoop Blast (10), HBase Word-Count (5), Building an Inverted Index (10), Build a Search Engine (20), Harp PageRank (10), Harp Mini-batch K-Means(20)]
Exams	(50%)	Midterm (20%), Final (30%)

1.14 Weekly Schedule

Week	Content	Section No	Assignment	Lab
1	• Course Info • Introduction • Data Center Model • Data Intensive Sciences • IaaS, PaaS, and SaaS • Challenges	Unit 1 (II)	Assignment 0	
2	• Course Expectations • Student Work 1 • Student Work 2 • Growth of Virtual Machines • Implementation Levels • Tools and Mechanisms • CPU, Memory and I/O Devices • Clusters and Resource Management • Data Center Automation • 'Distributed and Cloud Computing' Textbook: Chapter 1 (Sections 1.1, 1.2.4, 1.2.5, 1.3, 1.4.3)	Unit 2 (III)	Reading 1	Lab 0
3	• Apache Data Analysis Open Stack • MapReduce	Unit 3 (VI)	Project 1	Lab 1

4	• Hadoop Framework • Hadoop Tasks • Fault Tolerance • How Hadoop Runs on a MapReduce Job • Hadoop PageRank • “The Google File System”, published October 2003 • “MapReduce: Simplified Data Processing on Large Clusters”, published December 2004 • “BigTable: A Distributed Storage System for Structured Data”, published November 2006	Unit 3,Unit 4 (VI V)	Project 2	Lab 2
5	• Applying for FutureSystems Account • FutureSystems India OpenStack • Starting VMs on FutureSystems • Textbook: Chapter 3 (Sections 1-5). • Textbook: Chapter 4 (Sections, 1-6).	Unit 2 (III)		Lab Openstack
6	• Introduction to BLAST • BLAST Parallelization • "CloudBurst: highly sensitive read mapping with MapReduce" • "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications", IEEE eScience 2008 • "The Design, Implementation, and Evaluation of mpiBLAST", Bioinformatics Advance Access 2009 • "AzureBlast: A Case Study of Developing Science Applications on the Cloud", HPDC 2010 • "Applying Twister to Scientific Applications", CloudCom 2010	Unit 3 (VI)	Reading 2, Project 3	Lab 3
7	Midterm Exam			
8	• RDBMS vs. NoSQL • NoSQL Characteristics • BigTable • HBase • HBase Coding	Unit 5(VII)	Project 4	Lab 4

9	• Indexing Applications • Related Work • Indexamples • Indexing 101	Unit 5 (VII)	Project 5	Lab 5
11	• Google Components • Google Architecture • Google History	Unit 7 (VIII)	Project 6	Lab 6
12	• Iterative MapReduce Models • Parallel Processes • Static and Variable Data • MapReduce Model Comparison • Twister K-means • Coding and Iterative Alternatives • Textbook: Chapter 6 (Sections 2-5).	Unit 4 (V)	Project 7	Lab 7
13	• Everyday Data • Streaming the Data Ocean • Streams of Events • Faults & Frameworks • Spouts to Bolts • Textbook: Chapter 9 (Sections 1.4, 3, 4).	Unit 6 (IV)		Turtlebot and streaming
14	• Discussions and Parallel Thinking • Hadoop Extensions	Unit 4 (V)	Project 8	Lab 8
15	• Advanced Topics • SIMD vs. MIMD; SPMD vs. MPMD • Data Locality • Optimal Data Locality • Task Granularity • Resource Utilization and Speculative Execution	Unit 3 (VI)		Apache Spark and Flink
16	No new lecture assignment	Final Exam		

II

Cloud Computing Fundamentals

1.15 Overview

This section will change

The information in this section has been collected by Prof. Judy Qiu and taught at INdiana University. The first lecture introduces you to the schedule and homework submissions. Key cloud computing topics are highlighted. A selection of recommended and required textbooks is given, followed by an overview of the course structure. Please note that this section will change.

Overview (13:13) 

Overview (Page 1) 

Overview - pptx (Page 1) 

1.16 Introduction

Changes in computing technology for the past five decades are discussed. The rise of Big Data is shown in terms of its growth and significance. A prediction is made that the paradigm which has held 'til now of individual researchers with personal computers will give way to communities of researchers organizing through clouds. A more in-depth look at Unit 1 follows, focusing on the chapters from Distributed and Cloud Computing: From Parallel Processing to the Internet of Things.

Introduction (8:31) 

Introduction (currently unavailable) 

1.17 Data Center Model

A look at what truly defines a 'cloud'. Advantages like scalability and cost-effectiveness have promulgated commercial cloud offerings such as Amazon EC2. Cloud architecture as divided into three layers: Infrastructure as a Service, Platform as a Service, and Software as a Service. AzureBlast is used as an example of how to utilize the cloud setup. Certain misconceptions about clouds are then presented for further discussion.

Data Center Model (8:08) 

Data Center Model (currently unavailable) 

1.18 Data Intensive Sciences

Some time is spent analyzing the current age of vast data growth, where business, science, and consumer activity has seen an explosion of stored data measured in exobytes. In response to this, the way we conduct scientific research has also undergone an upgrade. However, the average scientist would rather focus on their own research rather than spend time trying to learn different methods of cloud and supercomputing.

Data Intensive Sciences (2:44) 

Data Intensive Sciences (currently unavailable) 

1.19 IaaS, PaaS and SaaS

Definitions and examples are given for Infrastructure as a Service, Platform as a Service, and Software as a Service. A chart is shown illustrating how use of clouds trades cost and control for efficiency. Following this is an exploration of the MapReduce program, and an illustration of its concepts through WordCount. Finally, four distinct approaches to MapReduce are compared.

IaaS/PaaS/SaaS (10:17) 

IaaS, PaaS and SaaS (currently unavailable) 

1.20 Challenges

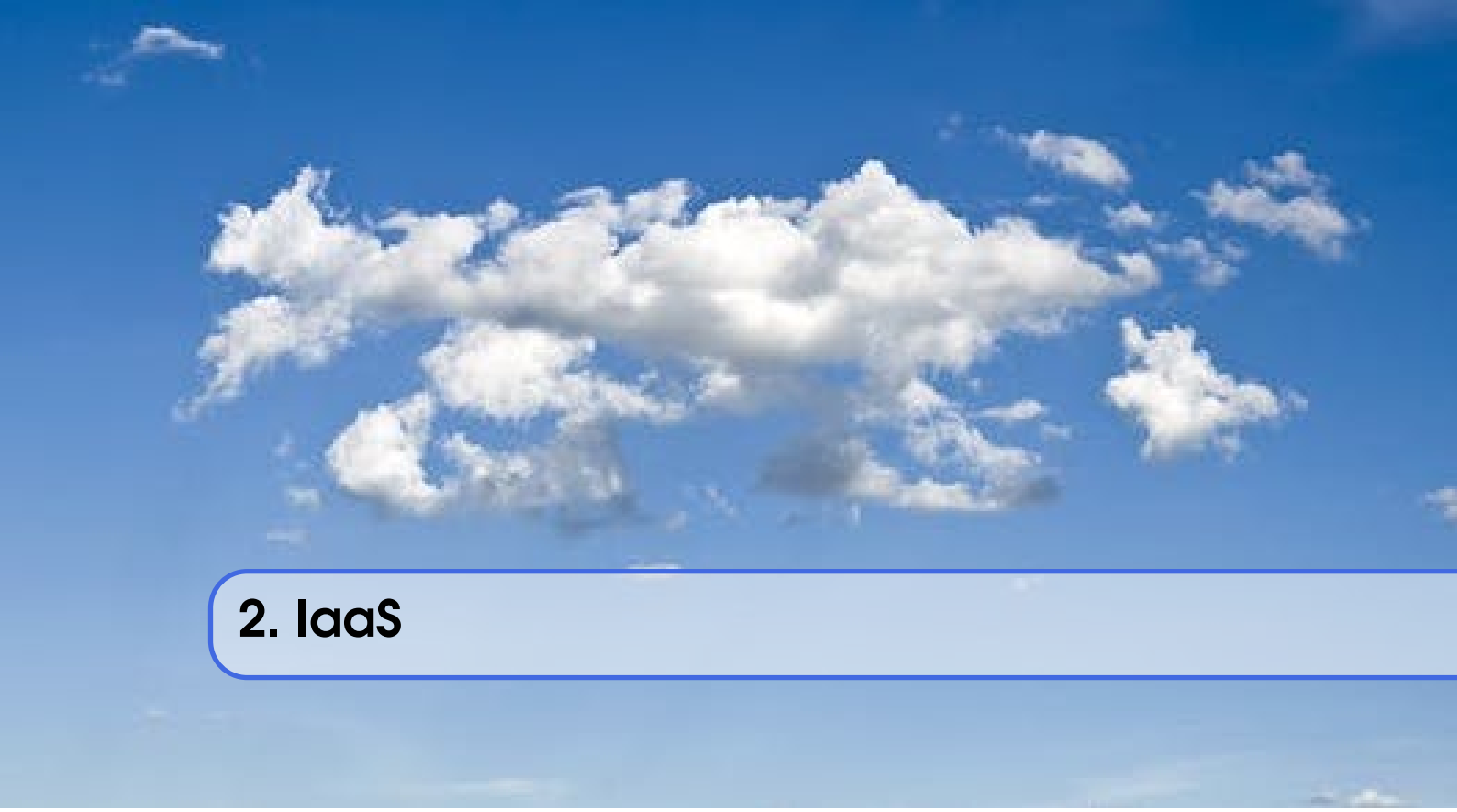
The demands of Big Data calls for advances in areas like distributed computing, systems management, internet technology, and hardware. Clouds have become more prominent in the last few decades, so much so that many people today take advantage of them without even knowing it. Of course, this has also led to increased concerns about security, price, tech support, etc. In spite of this, clouds still have clear advantages over traditional computing models. A quiz is offered at the end asking students to correctly place software in a hierarchy of computing.

Challenges (5:27) 

Challenges (currently unavailable) 

How to Run VMs (IaaS)

17 Video lectures (2 hours 7 minutes 9 seconds)



2. IaaS

2.1 Course Expectations

Examples and definitions are given for SaaS, PaaS, and IaaS. Computational models must be designed with the problems and effective resources in mind. A demonstration of cloud use for Bioinformatics on the FutureGrid educational testbed shows how clouds offer advantages of provisioning and virtual cluster support. Overhead and performance issues are touched upon through charts showing the use of three different virtual clusters.

Course Expectations (7:45)

Course Expectations (Page 1)

Course Expectations - pptx (Page 1)

2.2 Student Work 1

The work of previous B649 students is presented, covering a variety of topics and software: HBase, DryadLINQ, virtualization, commercial cloud storage, and IU's Twister iterative MapReduce.

Student Work 1 (8:48)

Student Work 1 (Page 7)

Student Work 1 - pptx (Page 7)

2.3 Student Work 2

This lecture offers further examples of prior student projects in areas like cloud storage, infrastructure and platforms, as well as high level languages like Apache Pig. Other topics cover MapReduce in Single Program Multiple Data mode, genetic algorithms in MapReduce, Hadoop supporting recommender systems, K-means clustering, etc. Following this the benefits/vulnerabilities of clouds and some of their most well-known programs are showcased in other research assignments.

Student Work 2 (10:03) 

Student Work 2 (Page 12) 

Student Work 2 - pptx (Page 12) 

2.4 Growth of Virtual Machines

Importance of virtualization is explored, including cross-platform applications. Virtualization has seen rapid growth in recent years in terms of use and services offered. Virtual machines differ from traditional computers in that software virtualization layer (hypervisor) runs on hardware, allowing guest OS to run on top of host OS. VMs can run independent of hardware specifications. Four different types of VM architecture, defined by the layer which the virtual machine monitor (VMM) runs on. VM is identical to physical machines and can be saved and stored, as well as migrated across hardware.

Growth of Virtual Machines (10:16) 

Growth of Virtual Machines (Page 28) 

Growth of Virtual Machines - pptx (Page 28) 

2.5 Implementation Levels

Virtualization can be implemented on five levels: application, library, OS, hardware, and instruction. Their benefits are compared in terms of performance, flexibility, complexity, and isolation. A layout is provided for the Linux virtualization layer, OpenVZ (OS level), which creates virtual private servers. CUDA is a high performance computing library, not designed for VMs; vCUDA is a virtual layer that allows interaction between CUDA and VMs, creating a virtual CUDA library.

Implementation Levels (7:57) 

Implementation Levels (Page 41) 

Implementation Levels - pptx (Page 41) 

2.6 Tools and Mechanisms

A list of major hypervisors is given. Type 1 hypervisor resides on the bare metal computer, while Type 2 runs over the host OS. XEN is an open source hardware level hypervisor: consists of hypervisor, kernel, and application. Domain0 in XEN is a VM that manages other VMs. Two types

of hardware virtualization: full virtualization and host-based virtualization. Para-virtualization does not need to modify the guest OS like full virtualization and works through hypercalls. An example is the ESX server from VMware.

Tools and Mechanisms (7:32) 

Tools and Mechanisms (Page 47)  *Tools and Mechanisms - pptx (Page 47)* 

2.7 CPU, Memory & I/O Devices

A hybrid approach to virtualization involves offloading some tasks to the hardware to reduce overhead. This can be combined with para-virtualization for even greater effects. In a guest OS, the VMM provides shadow page tables to transfer virtual memory to machine memory. An example is shown in the Intel Extended Page Table. A virtualization layer for an I/O device is possible, allowing it to act like a physical device and manage host and guest addresses, shown in a detailed VMware example.

CPU, Memory & I/O Devices (6:41) 

CPU, Memory & I/O Devices (Page 58)  *CPU, Memory & I/O Devices - pptx (Page 58)* 

2.8 Clusters and Resource Mgmt.

Characteristics of VM clusters are listed, including the ability to run multiple VMs on the same node and size alteration. Physical clusters are linked through nodes, while virtual clusters can be linked through physical or virtual nodes and can be replicated in virtual servers. Prepackaged OS can be installed in a virtual cluster. Should a VM fail for any reason, its image can be migrated to a new host so work is not lost. An example of this is demonstrated with XEN.

Clusters and Resource Management (5:07) 

Clusters and Resource Management (Page 66)  *Clusters and Resource Management - pptx (Page 66)* 

2.9 Data Center Automation

Whole data centers can be virtualized, enabling for the construction of private clouds. Some tools for Infrastructure as a Service clouds are Nimbus, Eucalyptus, OpenNebula, and vSphere. Eucalyptus is

shown in greater detail. Trust issues in cloud security are answered in virtual machines. Suggested reading material is provided at the end.

Data Center Automation (3:30) 

Data Center Automation (Page 74)  *Data Center Automation - pptx (Page 74)* 

2.10 Clouds in the Workplace

Clouds run as servers for data storage and sharing on the Internet in an on-demand capacity. Cloud services are scalable depending on the client's needs, allowing for a seemingly limitless source of computing power that can expand or shrink to meet financial demands. Some examples of cloud services are LinkedIn, Amazon S3, and Google App Engine. Different variations of clouds like IaaS and PaaS are offered by both open source and commercial providers. Cloud systems are composed of separate elements like Eucalyptus, Xen and VMware.

Clouds in the Workplace (7:13)  *Clouds in the Workplace (currently unavailable)* 

2.11 Checklists & Challenges

The capabilities of several IaaS cloud structures like Amazon EC2 or PaaS like Microsoft Azure are listed. Public and private clouds share certain features; the main difference is public clouds are owned by service providers while private clouds are offered by individual corporations. Certain enabling technologies are required for clouds to provide quick and scalable computing. These include virtual cluster provisioning and multi-tenant environments. PaaS demands the capability to process huge amounts of data as in the case of web searches. Some challenges faced by cloud computing include vendor lock-in owing to lack of standard APIs and metrics; for scientists, there is uncertainty about whether experiments can be reproduced effectively in different cloud environments. However there are distinct advantages clouds potentially have to offer: standardized APIs can eliminate lock-in, and encryption offers data confidentiality.

Checklists & Challenges (9:08)  *Checklists & Challenges (currently unavailable)* 

2.12 Data Center Setup

Huge data centers enable cloud computing, containing up to a million servers. Large data centers charge less for their services than small ones. A diagram illustrates the typical setup of a cloud; rack space on the bottom, on top of which are load balancers, then excess routers and border routers. The next figure compares cost effectiveness in a traditional IT model to a cloud. Other figures display small server clusters and a typical data center arrangement, including emergency power supply and cooling system. A chart shows the power consumption based on CPU, disk, etc. Disks in warehouse servers may be onsite or attached to outside connections like InfiniBand. Switches can form an array of racks. The distribution of memory across a local, rack, or array server in warehouse server setup is listed.

Data Center Setup (7:49) *Data Center Setup (currently unavailable)*

2.13 Cultivating Clouds

Power utilization effectiveness (PUE) for a warehouse is determined by comparing it to IT power usage. Racks can contain 40 servers, shipping containers can have up to 1,000 servers; a data center could take 2 years to construct. Warehouse scale computing has greater economy of scale than data centers by reducing network and administrative costs. Individual users can interact with clouds in the SaaS model, while organizations use PaaS. Clouds generally use VMs to recover from system failures. It is predicted that the cloud job market and demand for clouds will experience great growth in the future. Clouds have become ubiquitous in all aspects of the private and public sector. In the future clouds must take into account user privacy, data security and copyright protection.

Cultivating Clouds (5:10) *Cultivating Clouds (currently unavailable)*

2.14 Applying for FutureSystems Account

Applying for FutureSystems Account (5:32)

Cultivating Clouds (Page 1) *Cultivating Clouds - pptx (Page 1)*

2.15 FutureSystems India OpenStack

FutureSystems India OpenStack (10:28)

FutureSystems India OpenStack (Page 10) *FutureSystems India OpenStack - pptx (Page 10)*

2.16 Starting VMs on FutureSystems

Starting VMs on FutureSystems (6:40)

2.17 Hadoop WordCount on VMs

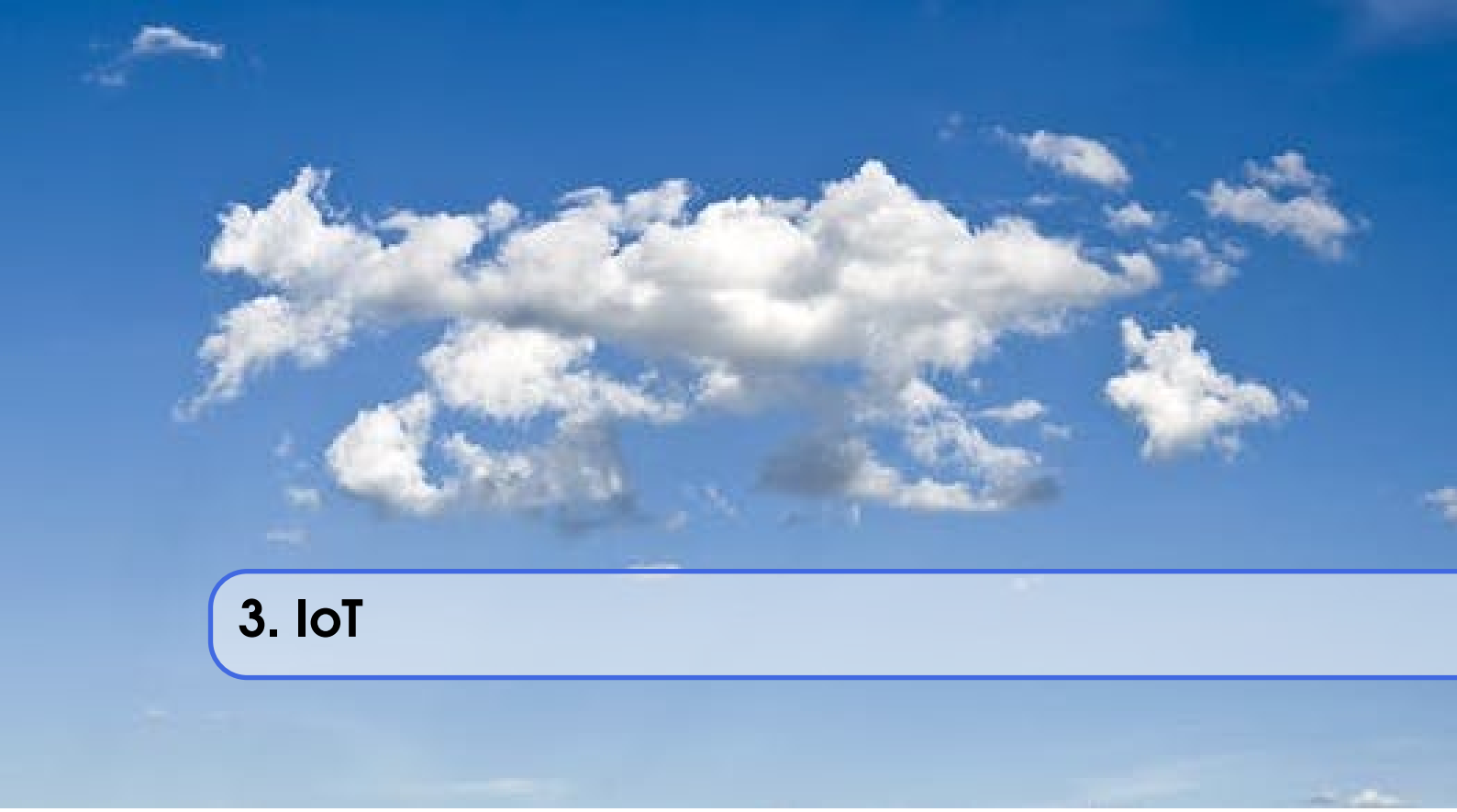
Hadoop WordCount on VMs (7:30)


Hadoop WordCount on VMs (Page 17)

Hadoop WordCount on VMs - pptx (Page 17)

IV Internet of Things

5 Video lectures (46 minutes 21 seconds)



3. IoT

3.1 Everyday Data

Ph.D. candidate Supun Kamburugamuva goes over the so-called Internet of Things as well as strategies and tools developed for Distributed Stream Processing.

Everday Data (9:31) *Everyday Data (currently unavailable)*

3.2 Streaming the Data Ocean

Ph.D. candidate Supun Kamburugamuva goes over the so-called Internet of Things as well as strategies and tools developed for Distributed Stream Processing.

Streaming the Data Ocean (9:38) *Streaming the Data Ocean (currently unavailable)*

3.3 Streams of Events

Ph.D. candidate Supun Kamburugamuva goes over the so-called Internet of Things as well as strategies and tools developed for Distributed Stream Processing.

Streams of Events (10:44) *Streams of Events (currently unavailable)*

3.4 Faults & Frameworks

Ph.D. candidate Supun Kamburugamuva goes over the so-called Internet of Things as well as strategies and tools developed for Distributed Stream Processing.

Faults & Frameworks (7:46)  *Faults & Frameworks (currently unavailable)* 

3.5 Spouts to Bolts

Ph.D. candidate Supun Kamburugamuva goes over the so-called Internet of Things as well as strategies and tools developed for Distributed Stream Processing.

Spouts to Bolts (8:42)  *Spouts to Bolts (currently unavailable)* 

V

How to Run Iterative MapReduce (PaaS)

12 Video lectures (1 hour 37 minutes 32 seconds)

4. Itterative Map Reduce

4.1 MapReduce Refresher

A review covers cloud computing levels, MapReduce, the course structure, etc. This is followed by a look at Google and their initial offering, Google search engine. Amount of tasks performed on this engine increased considerably over the course of a single decade.

MapReduce Refresher (9:00) *MapReduce Refresher (currently unavailable)*

4.2 Google Search Engine 1

The Google web server relies on index and doc servers. Index servers allow the search engine to not have to depend on manually checking every document, reducing computing power demands. Index partitioning can be accomplished either through subsets of documents or words. Basic differences between index and doc servers are discussed. Cache servers save previous query results and can bypass index/doc servers for repeat queries.

Google Search Engine 1 (8:04)

Google Search Engine 1 (Page 15) *Google Search Engine 1 - pptx (Page 15)*

4.3 Google Search Engine 2

Cache servers greatly enhance the performance of search engines. However, this duplication of queries can lead to higher latency. Crawling in a search engine handles subsets of websites. Batch indexing is the simplest way to create indexes, although it lacks advanced features like

checkpointing, which could lead to issues down the line. In-memory index added to Google over a decade ago; increases throughput and decreases latency. Image-based and video-based searches were added in 2007, among others. Google File System, MapReduce and BigTable are key components of Google's current search structure. A discussion of the initial Google proposal paper follows.

Google Search Engine 2 (8:32)

Google Search Engine 2 (Page 21) *Google Search Engine 2 - pptx (Page 21)*

4.4 Hadoop PageRank

PageRank algorithm in Google ranks a webpage's popularity and relevance. The PageRank calculation formula is examined. After this comes an example of its performance and further mathematical formulae involved in its application.

Hadoop PageRank (7:58)

Hadoop PageRank (Page 1) *Hadoop PageRank - pptx (Page 1)*

4.5 Discussions and ParallelThinking

Four types of MapReduce: pleasingly parallel, classic, iterative, and loosely synchronous. A diagram shows the flow of data in MapReduce. Specific formulae for PageRank are shown with and without the damping factor. Key-value pairs can be written in matrix form by defining the keys as nodes. Map tasks must make sure to handle dangling nodes (isolated from neighbors), distributed page-rank contribution, and reducer output being the same format as map input. Reduce input is key-value pairs. Ideas behind parallel thinking are analyzed, along with a list of related reading. Seven important questions are asked concerning parallel computing. 13 'Dwarves' are different methods of parallel computing, including MapReduce.

Discussions and ParallelThinking (11:12)

Discussions and ParallelThinking (Page 10) *Discussions and ParallelThinking - pptx (Page 10)*

4.6 Hadoop Extensions

A model of MapReduce shows its structure. Dryad is Microsoft's version of parallel processing. Twister is an iterative map-reduce framework, as are Haloop, Spark and Pregel. A comparison of their features and capabilities is included.

Hadoop Extensions (5:37)  *Hadoop Extensions (currently unavailable)* 

4.7 Iterative MapReduce Models

An introduction to the idea of iterative MapReduce. An overview of other MapReduce models follows. Map Only model has parallel map tasks with no communication between them. Classic MapReduce involves parallel map tasks and reduce tasks which aggregate output and allow legacy code. Loosely Synchronous is an MPI model used in computation and communication of scientific applications.

Iterative MapReduce Models (6:46) 

Iterative MapReduce Models (Page 1)  *Iterative MapReduce Models - pptx (Page 1)* 

4.8 Parallel Processes

CPU performance increases according to Moore's Law can no longer keep up with the high volume of data being generated. Multi-core architecture is a response to this issue. It requires runtime approaches supporting parallelism, either data-centric for higher throughput (MapReduce) or the traditional HPC approach for optimized computation performance (MPI). MapReduce allows for moving computation to the data. A diagram illustrates the base MapReduce process. MapReduce is designed to improve I/O and handle intermediate data, task scheduling, and fault tolerance. Versions of MapReduce like Hadoop, Dryad and MPI boast different features and programming languages.

Parallel Processes (9:44) 

Parallel Processes (Page 4)  *Parallel Processes - pptx (Page 4)* 

4.9 Static and Variable Data

Iterative MapReduce was introduced to support high performance systems. It runs iterations of the map/reduce cycles. Data mining algorithms like K-means run numerous iterations. Static data such as data points in K-means does not change, while variable data can alter between each iteration. A naïve iterative MapReduce model can generate huge overhead owing to constantly referencing static data. This can be overcome with long-running map/reduce tasks that distinguish between static and variable data. You can also accelerate the intermediate data transfer or combine the output of all reduce tasks. Iterative MapReduce is shown in the Twister program, which uses the combine output method and determines at the end of every iteration whether to stop or continue with further iterations. The master node in Twister is the Twister Driver, and the slave nodes are Twister Daemons. Twister stores I/O data in partition files. Three MapReduce patterns in Twister: 1) Large input data, reduced in the end; 2) Data size is constant; 3) Data volume increases after MapReduce execution. Data Manipulation Tool handles data loading and uses metadata to keep track of data in partitions. Twister employs static scheduling. Fault tolerance is reserved for failures

that terminate running tasks. Static data can then be used to reassign the failed iterations. A list of Twister APIs is given.

Static and Variable Data (11:01)

Static and Variable Data (Page 10) *Static and Variable Data - pptx (Page 10)*

4.10 MapReduce Model Comparison

This video showcases examples of work done comparing Twister results with Hadoop, MPI and DryadLINQ. The first is Map Only with CAP3 DNA Sequence Assembly, followed by Classic MapReduce with Pair-wise Sequences and High-Energy Physics, Iterative with K-means clustering, PageRank and Multi-dimensional Scaling, and finally Loosely Synchronous with Matrix Multiplication Algorithms. In all cases, Twister outperforms or is close to the competition.

MapReduce Model Comparison (6:56)

MapReduce Model Comparison (Page 24) *MapReduce Model Comparison - pptx (Page 24)*

4.11 Twister K-means

Twister is applied to K-means Clustering. K-means develops a set number of clusters by creating cluster centers (centroids) that encompass the data points after successive proximity calculations. Parallelization of K-means is accomplished in the partitions, and the final centroids are determined in the Reduce step. A sample of K-means Clustering code follows, after which Twister is shown being used to determine centroids on K-means. Several questions are posed pertaining to the features of Twister. The results of a Twister K-means run are compared with those from a sequential run. Shown here, as the number of data points increases, Twister's runtimes get progressively faster. In a final set of runs against Hadoop, DryadLINQ, and MPI, Twister outperforms all but MPI.

Twister K-means (7:28)

Twister K-means (Page 34) *Twister K-means - pptx (Page 34)*

4.12 Coding and Iterative Alternatives

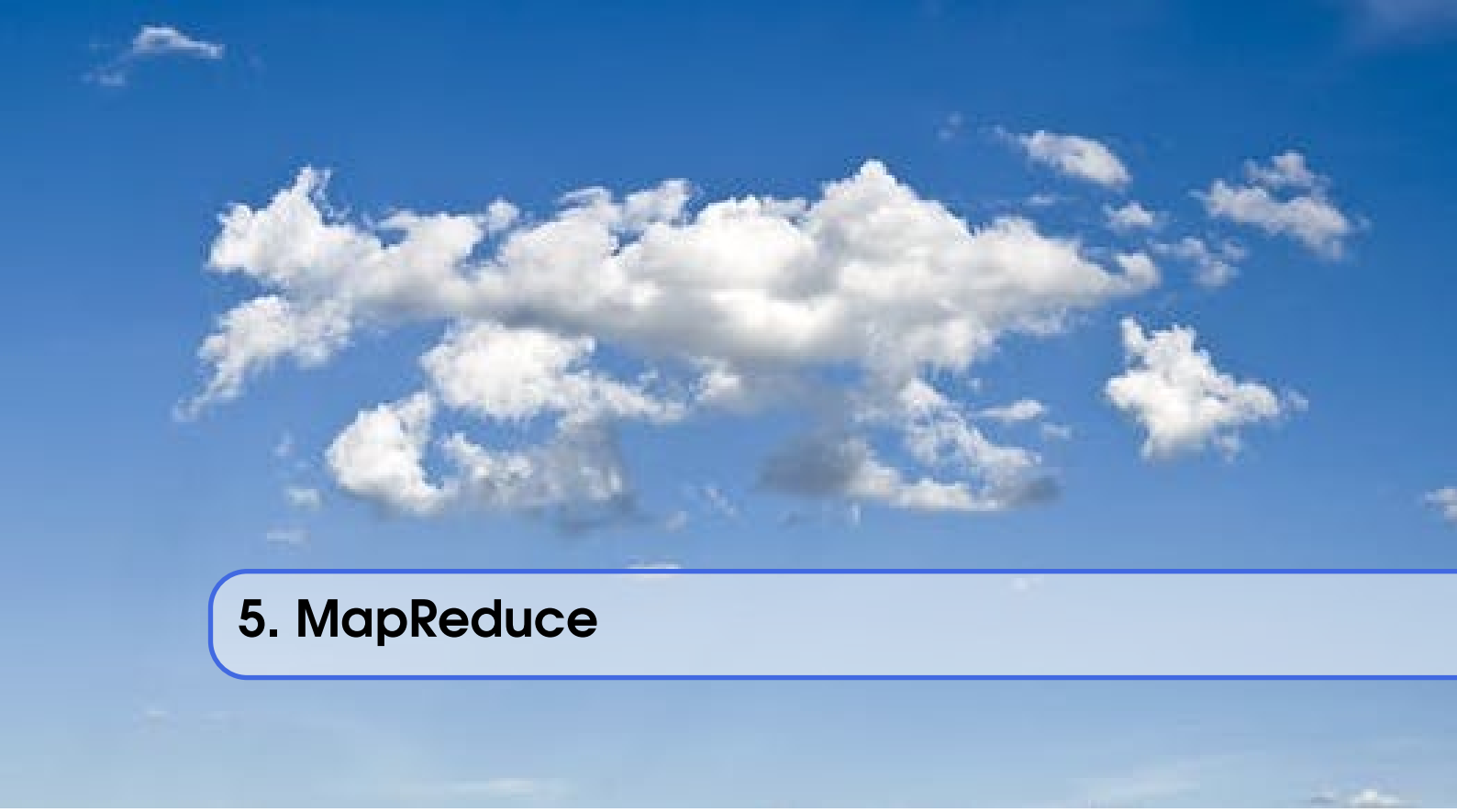
A more detailed look is taken at the code used to run Twister K-means. MapReduce has many programs designed around its setup, including other iterative versions like Haloop, Pregel, and Spark. Twister can extend the use of traditional MapReduce to more complex applications.

Coding and Iterative Alternatives (5:14) 

Coding and Iterative Alternatives (Page 43)  *Coding and Iterative Alternatives - pptx (Page 43)* 

VI How to Run MapReduce (PaaS)

15 Video lectures (1 hour 58 minutes 4 seconds)



5. MapReduce

5.1 Apache Data Analysis OpenStack

The buildup of Big Data has seen the development of new data storage systems like MapReduce and Hadoop. Apache's Big Data Stack houses a host of programs designed around Google's offerings like MapReduce. The architecture of Hadoop 1.0 and 2.0 are compared, along with an examination of the MapReduce concept. A demo video of Twister-MDS includes a 3-dimensional representation of data cluster sorting through the PlotViz program. Data analysis tool Twister boasts features like in-memory support of tasks, data flow separation, and portability.

Apache Data Analysis OpenStack (12:01)

Apache Data Analysis OpenStack (Page 1) *Apache Data Analysis OpenStack - pptx (Page 1)*

5.2 MapReduce

MapReduce was designed by Google to address the problem of large-scale data processing. A breakdown of basic MapReduce terms and functions follows. Use of MapReduce has flourished since its premier, as illustrated by an in-depth example of its use in WordCount. Finally the basic process of MapReduce is shown.

MapReduce (9:07)

MapReduce (Page 6) 

MapReduce - pptx (Page 6) 

5.3 Hadoop Framework

Hadoop is an open source version of MapReduce designed for broad application in terms of code and settings. Storage is done in the Hadoop Distributed File System through master and slave nodes. Compute is handled by JobTracker and TaskTracker; the duties of these two intertwined programs are then explored more fully.

Hadoop Framework (8:32) 

Hadoop Framework (Page 15) 

Hadoop Framework - pptx (Page 15) 

5.4 Hadoop Tasks

The Map stage of MapReduce is shown in greater detail. This process starts with Hadoop Distributed File System, which handles the input data. Key value pairs are assigned to the data blocks. Combiner reduces data size and Partitioner determines distribution of keys among reducers. Intermediate data is stored in a circular buffer before being sent to reduce tasks. Shuffle and Merge are used to order and reduce size of intermediate data. Reduce tasks take over then to determine the output data format. A final chart illustrates the concept of parallelism in MapReduce.

Hadoop Tasks (11:01) 

Hadoop Tasks (Page 24) 

Hadoop Tasks - pptx (Page 24) 

5.5 Fault Tolerance

Fault tolerance is a natural benefit of MapReduce. The master node pings worker nodes regularly to verify they are working, and acts accordingly if they do not respond. A diagram illustrates the files which are in charge of things like number of map and reduce tasks, and what to do when the limit is reached on the buffer. The lecture ends with a discussion of class assignments.

Fault Tolerance (2:45) 

Fault Tolerance (Page 36) 

Fault Tolerance - pptx (Page 36) 

5.6 Programming on a ComputerCluster

Hadoop is now a large part of Yahoo!'s system setup, as well as handling a tremendous variety of data in other areas like medicine and business. A list of time spans for actions in system

requirements is given. The original MapReduce was designed to resolve problems like load balancing and machine failures.

Programming on a ComputerCluster (6:01) 

Programming on a ComputerCluster (Page 1)  *Programming on a ComputerCluster - pptx (Page 1)* 

5.7 How Hadoop Runs on a MapReduceJob

A detailed diagram of the MapReduce job framework is given. This includes task status updates, shuffling, and writing data to nodes. MapReduce is a C++ framework, while Hadoop is written in Java. Shuffling and sorting occurs in the map phase. Reduce reads and writes files to HDFS, and the merger generates the final result. The second Quiz is given at the end.

How Hadoop Runs on a MapReduceJob (9:25) 

How Hadoop Runs on a MapReduceJob (Page 8)  *How Hadoop Runs on a MapReduceJob - pptx (Page 8)* 

5.8 Literature Review

This video deals primarily with scientific papers written on the topic of MapReduce and related programs. There is a certain criteria for judging scientific submissions. The first paper highlights Google File System, covering topics like data chunks, metadata, and replicas. This is followed by MapReduce and BigTable.

Literature Review (9:43) 

Literature Review (Page 16)  *Literature Review - pptx (Page 16)* 

5.9 Introduction to BLAST

There are four types of programming model communication patterns: embarrassingly parallel (only map), classic map/reduce, iterative map/reduce, and loosely synchronous. The basic bioinformatics BLAST (Basic Local Alignment Sequence Tool) program data flow is illustrated. An example of database creation comes from the Seattle Children's Hospital. BLAST uses scores to find similar sequences in databases.

Introduction to BLAST (8:27)

Introduction to BLAST (Page 1) *Introduction to BLAST - pptx (Page 1)*

5.10 BLAST Parallelization

The role of master and worker nodes in BLAST multi-thread usage is discussed. BLAST can be parallelized in several ways: multi-thread, query segmentation, and database segmentation. BLAST is pleasingly parallel in application, but many programs are not. Further information about articles featuring BLAST is provided at the end.

BLAST Parallelization (4:44)

BLAST Parallelization (Page 13) *BLAST Parallelization - pptx (Page 13)*

5.11 SIMD vs MIMD;SPMD vs MPMD

Four types of parallel models: SISD (traditional PCs), SIMD (GPUs), MISD (shuttle flight control computer), MIMD (distributed systems). Point-to-point (P2P) communication in MPI is used as an example of parallelization. Each successive process adds its own stamp to the data before passing it on to the next. Matrix multiplication for scientific applications differs from the norm in that data is sent in a matrix, not a string. WordCount functions in a map/reduce pattern. These are all types of SIMD. SPMD and MPMD are two other types of model.

SIMD vs MIMD;SPMD vs MPMD (9:42)

SIMD vs MIMD;SPMD vs MPMD (Page 1) *SIMD vs MIMD;SPMD vs MPMD - pptx (Page 1)*

5.12 Data Locality

A brief review is given of previous topics. As opposed to MPI and HPC, MapReduce brings the computation to the data, rather than vice-versa. This is done to limit energy usage and network congestion. Several factors such as number of nodes and tasks can impact data locality. An equation to improve data locality is tested in an experiment, whose results are given. By default, Hadoop determines scheduling of tasks to available slots in terms of best local composition, not global.

Data Locality (8:36)

Data Locality (Page 10) *Data Locality - pptx (Page 10)*

5.13 Optimal Data Locality

Global data optimization can be achieved through a proposed algorithm given here. Task, slot, and cost are factors in this algorithm. Network bandwidth must also be taken into consideration when assigning tasks to slots. Linear Sum Assignment Problems require greater time to finish when matrix size is increased. Two different scheduling algorithms were designed to improve the original one in Hadoop. An experiment was run comparing all three, with the network topology-aware algorithm clearly outperforming the others.

Optimal Data Locality (4:17) 

Optimal Data Locality (Page 17) 

Optimal Data Locality - pptx (Page 17) 

5.14 Task Granularity

Size of data blocks affects load balancing and overhead. Using Bag of Divisible Tasks method, tasks can be split into sub-tasks and distributed amongst slots to maximize efficiency. When splitting tasks, one must take into account when and which tasks to split, as well as how and how many. In our current proposed algorithm, tasks are split until each slot is occupied. It also uses ASPK (Aggressive Scheduling with Prior Knowledge) to split larger tasks first and when the performance gain is deemed optimal. Optimal and Expected Remaining Job Execution Time can help determine task splitting. Several examples are offered with either single or multiple jobs.

Task Granularity (9:51) 

Task Granularity (Page 29) 

Task Granularity - pptx (Page 29) 

5.15 Resource Utilization and Speculative Execution

Resource stealing involves appropriating cores that are kept in reserve on separate nodes and returning them when the computation is over. Speculative execution addresses fault tolerance; when the master node notices a task is running slowly, it will start a speculative task which can take over if it is determined the original task will not finish in time. Overuse of speculative tasks can lead to poor data locality and higher energy demands.

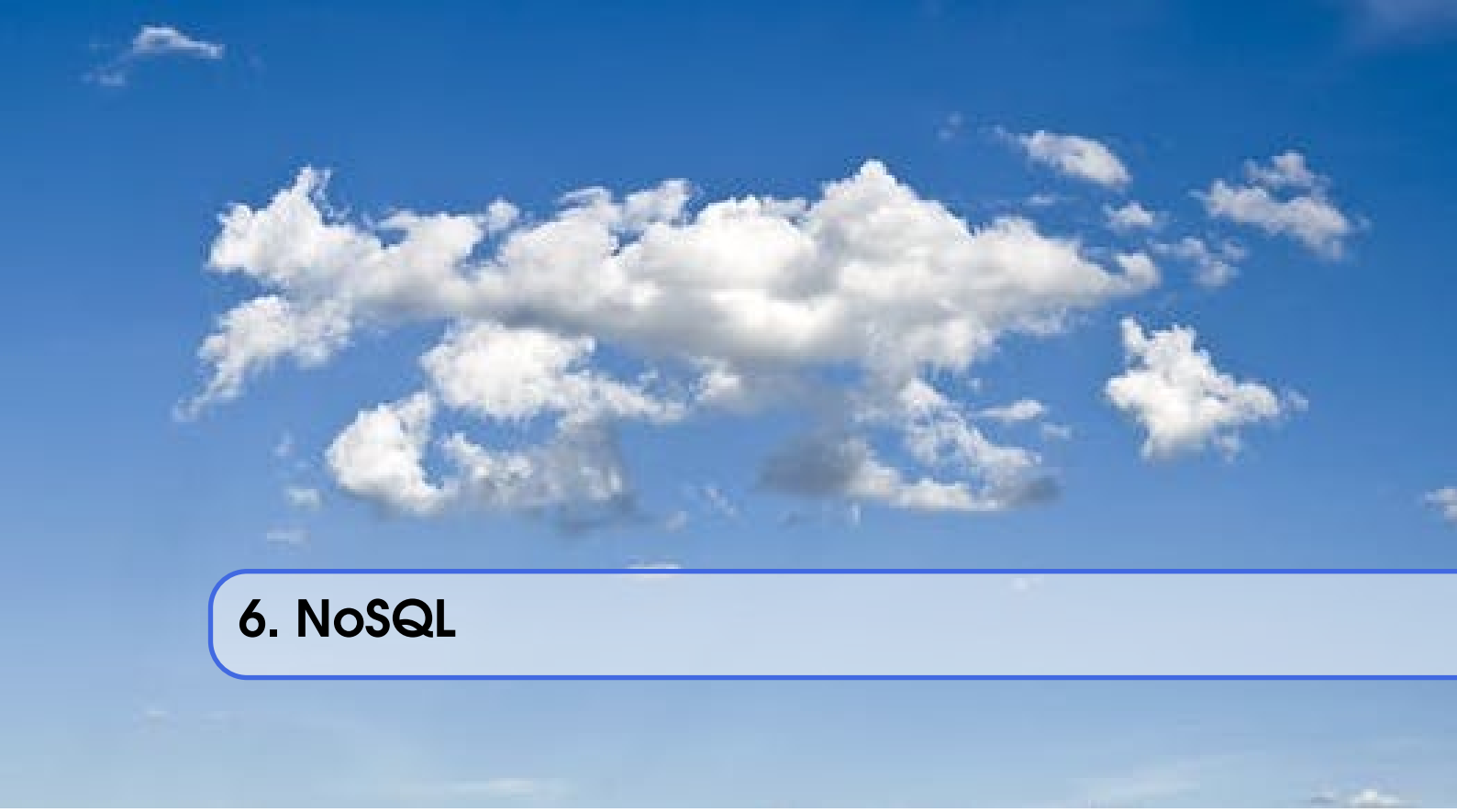
Resource Utilization and Speculative Execution (3:52) 

Resource Utilization and Speculative Execution (Page 46) 

Resource Utilization and Speculative Execution - pptx (Page 46) 

VII How to Store Data (NoSQL)

11 Video lectures (1 hour 26 minutes 8 seconds)



6. NoSQL

6.1 RDBMS vs. NoSQL

RDBMS vs. NoSQL (9:22)

RDBMS vs. NoSQL (Page 1)

RDBMS vs. NoSQL - pptx (Page 1)

6.2 NoSQL Characteristics

Clouds have arisen as an answer to the data demands of social media. Three major programs for NoSQL are BigTable, Dynamo, and CAP theory. NoSQL is not meant to replace SQL, but to tackle the large-data problems SQL is not well equipped to handle. SQL ACID transactions are Atomic, Consistent, Isolated, and Durable. Consistency can be either strong (ACID) or weak (BASE). CAP theorem offers Consistency, Availability, and Partition tolerance, only two of which can coexist for a shared-data system. NoSQL comes in two varieties, each with pros and cons: Key-Value or schema-less. Common advantages of NoSQL include their being open source and fault tolerant.

NoSQL Characteristics (10:31)

NoSQL Characteristics (Page 11)

NoSQL Characteristics - pptx (Page 11)

6.3 BigTable

Big Table is a key-value NoSQL model with data arranged in rows and columns. It is composed of Data File System, Chubby, and SSTable. A tablet is a range of rows in BigTable. The master

node assigns tablets to tablet servers and manages these servers. Memory is conserved by making SSTables and memtables compact. BigTable is used in features of Google like their search engine and Google Earth.

BigTable (6:55) 

BigTable (Page 28) 

BigTable - pptx (Page 28) 

6.4 HBase

HBase is a NoSQL core component of the Hadoop Distributed File System. It is a scalable distributed data store. A timeline of HBase and Hadoop is shown. BigTable still has its uses but does not scale well to large amounts of analytic processing. HBase has a row-column structure similar to BigTable as well as master and slave nodes. Its place in the architecture of HDFS is shown in a diagram.

HBase (7:37) 

HBase (Page 44) 

HBase - pptx (Page 44) 

6.5 HBase Coding

This video gives an overview of the code used in the installation of HBase and connecting to it.

4:30 (HBase Coding) 

HBase Coding (Page 60) 

HBase Coding - pptx (Page 60) 

6.6 Indexing Applications

A brief summary of the course up to this point is given, followed by a diagram showing the setup of a search engine. Google's search engine contains three key technologies: Google File System, BigTable, and MapReduce. However, research into big data remains difficult owing to the scope of its size. Social media data in particular is a huge source of data with numerous subsets, all of which demands specific approaches in terms of search queries. There are three stages to this approach: query, analysis, and visualization.

Indexing Applications (9:33) 

Indexing Applications (Page 1) 

Indexing Applications - pptx (Page 1) 

6.7 Related Work

Indexing improves efficiency in querying data subsets and analysis. Indices can be single (B+, Hash) or multi-dimensional (R, Quad). Four databases which utilize indexing are HBase, Cassandra, Riak, and MongoDB. Current indexing strategies have limits; for instance, they cannot support range queries or only retrieve Top 'n' most relevant topics. Customizability of indexing among NoSQL databases is desirable.

Related Work (5:56) 

Related Work (Page 11) 

Related Work - pptx (Page 11) 

6.8 Indexamples

Mapping between metadata and raw index data is the essential issue with indexing. Examples are shown for HBase, Riak, and MongoDB. An abstract index structure contains index keys, entry IDs among multiple entries, and additional fields. Index configuration allows for customizability through choice of fields, which can be anything from timestamps, text, or retweet status.

Indexamples (8:35) 

Indexamples (Page 15) 

Indexamples - pptx (Page 15) 

6.9 Indexing 101

User-defined index allows a user to select the fields used in their search. Data records are indexed or un-indexed. Index structure is made up of key, entry ID, and entry fields. A walk-through customized index creation is shown on HBase, called IndexedHBase. HBase is suited to accommodate the creation of index tables. A performance test of IndexedHBase is done on the Truthy Twitter repository, displaying the various tables that can be created with different criteria. Loading time for large-scale historical data can be reduced by adding nodes. Streaming data can be handled by increasing loaders. A comparison of query evaluation is made between IndexedHBase and Riak, with Riak being more efficient with small data loads but IndexedHBase proving superior for large-scale data.

Indexing 101 (9:53) 

Indexing 101 (Page 20) 

Indexing 101 - pptx (Page 20) 

6.10 Social Media Searches

The Truthy Project archives social media data by way of metadata memes. Some problems faced in analyzing this data include its large volume, sparsity of information in tweets, and attempting to

arrange streaming tweets. Apache Open Stack upgrades Hadoop 2.0 with YARN and a new HDFS. A diagram displays an indexing setup for social media data with YARN.

Social Media Searches (6:19) 

Social Media Searches (Page 28) 

Social Media Searches - pptx (Page 28) 

6.11 Analysis Algorithms

Another method of use for inverted indices is in analysis algorithms. The mathematics involved in this is explored, as well as how it relates to index data, mapping, and reducing. Rather than scanning all raw data present, indices allow for searching only the relevant data. An example is given illustrating how this decreases the time needed to search hashtags in Twitter.

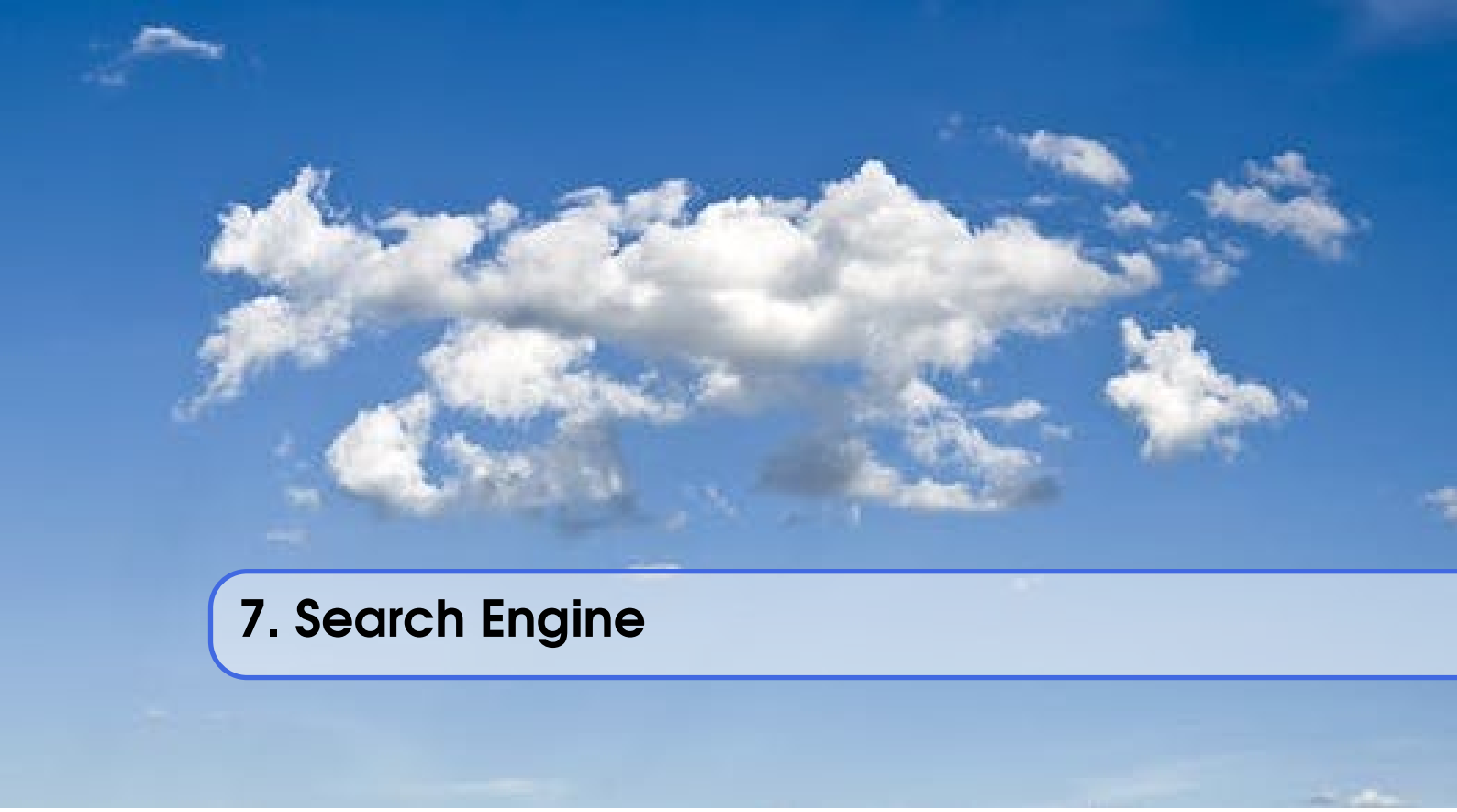
Analysis Algorithms (6:57) 

Analysis Algorithms (Page 35) 

Analysis Algorithms - pptx (Page 35) 

VIII How to Build a Search Engine(SaaS)

3 Video lectures (26 minutes 18 seconds)



7. Search Engine

7.1 Google Components

Google Components (7:02)

Google Components (Page 1)

Google Components - pptx (Page 1)

7.2 Google Architecture

Google Architecture (8:40)

Google Architecture (Page 6)

Google Architecture - pptx (Page 6)

7.3 Google History

Google History: https://youtu.be/Kg0NK0XUkHw?t=175 (starting 2:55)

Google Search Engine 1: https://www.youtube.com/watch?v=S2oT7uMw5Yg

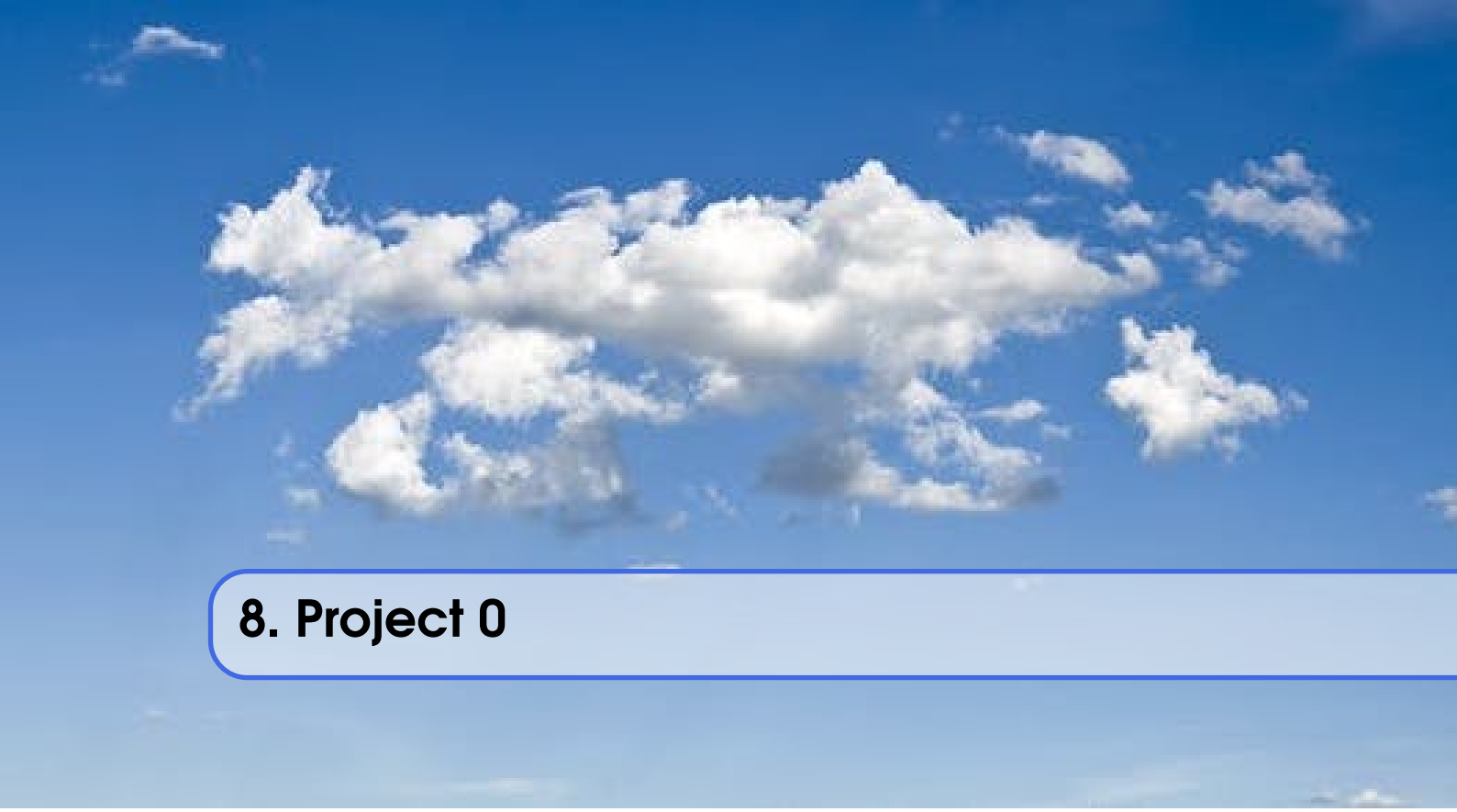
Google Search Engine 2: https://www.youtube.com/watch?v=pxos3Yt6y6I

Google History (10:36)

Google History (Page 14)

Google History - pptx (Page 14)

IX Assignments



8. Project 0

Project 0 will require you to identify the different software tools/technologies included in the given attachment and then group them into the correct layer of categories as indicated on the left-hand side of the slide.

This homework is worth 5 points.

5 points (Project 0)

9. Project 1

You will need to complete the source code and write a report. Zip your work into a file with the name username_project1.zip (replace 'username' with your Group Contact member's username) and submit the following:

- Complete source code
- **A document with the following details:**
  - Transformation of data during the computations, i.e. data type of key, value
  - The data structure used to transfer between Map and Reduce phases
  - How the data flow happens through disk and memory during the computation

Only one submission per group is required for Project 1.

Input Data (Project 0)

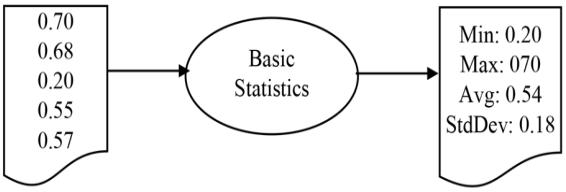
9.1 Project 1: Basic Statistics with Hadoop Cloud Computing Spring 2017

Judy Qiu

Problem statement

The idea of this project is to get you started with Hadoop and the MapReduce concept. You may have already looked at the WordCount example, both serial and Hadoop implementations. This problem is similar to WordCount except that you will be computing the basic statistics such as min, max, average, and standard deviation of a given data set.

The input to the program will be a text file carrying exactly one floating point number per line. The output should include **min, max, average, and standard deviation** of these numbers.



Files

A test input file is available as a separate attachment. The statistics values for this input are **Min: 0.01 Max: 0.99 Avg: 0.50 StdDev: 0.2817**

Deliverables

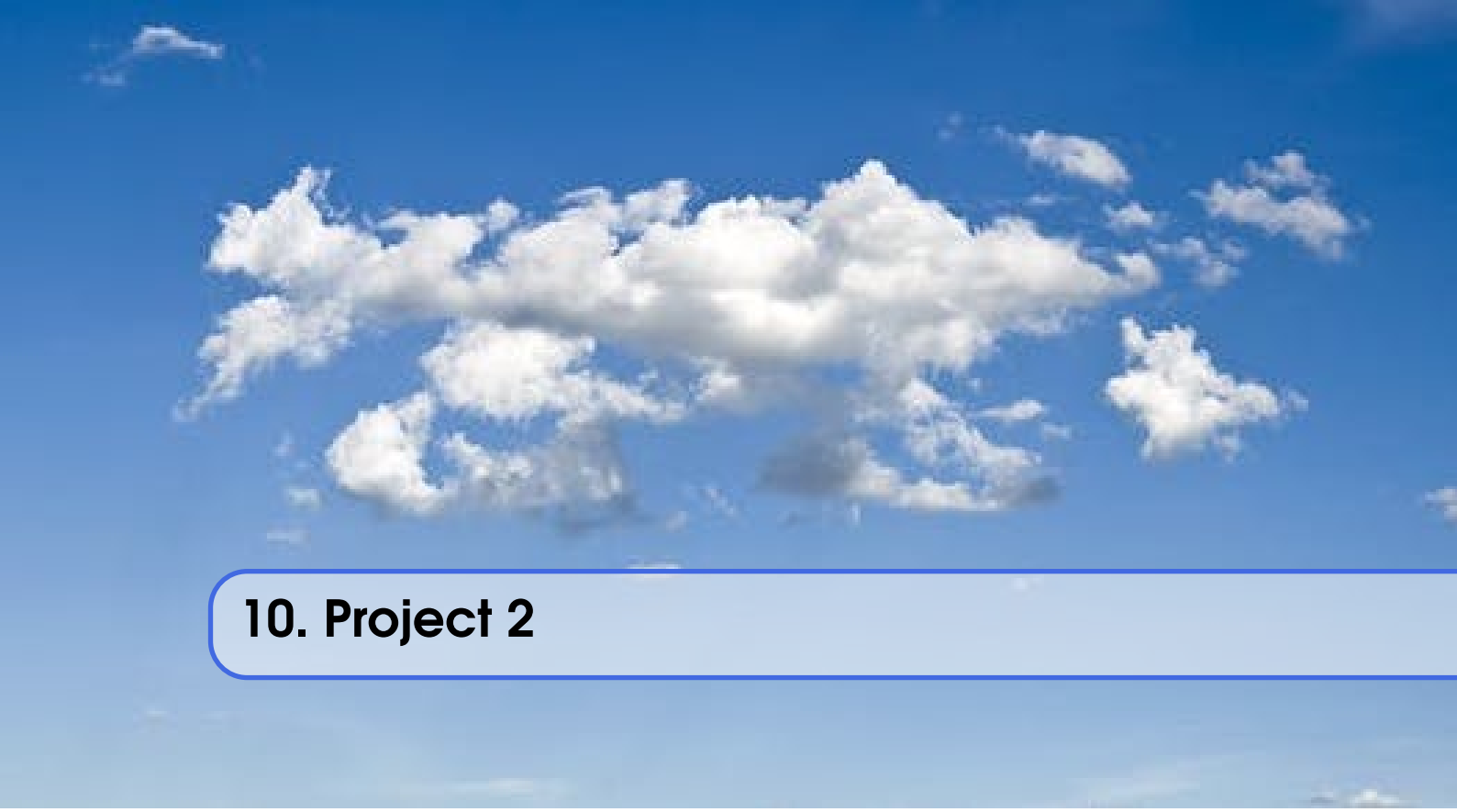
You will need to complete the source code and write a report. Zip your work into a file with the name username_project1.zip (replace 'username' with your own) and submit the following:

- Complete source code
- A document with the following details:
  - Transformation of data during the computations, i.e. data type of key, value
  - The data structure used to transfer between Map and Reduce phases
  - How the data flow happens through disk and memory during the computation

Evaluation

The point total for this project is 5.

- Correctness of the source code (2 points)
- Completeness of the report (3 points)



10. Project 2

You are required to turn in the following items in a zip file (username_HadoopPageRank.zip) in this assignment:

- The source code of Hadoop PageRank you implemented.
- **Technical report (username_HadoopPageRank_report.docx) that contains:**
  - The description of the main steps and data flow in your program.
  - The output file (username_HadoopPageRank_output.txt) which contains the first 10 urls along with their ranks.

Project 2: Hadoop PageRank
Cloud Computing
Spring 2017

Professor Judy Qiu

Goal

This assignment provides an illustration of PageRank algorithms and Hadoop. You will then blend these applications by implementing a parallel version of PageRank using the programming interfaces of the Hadoop MapReduce framework.

Deliverables

You are required to turn in the following items in a zip file (username_HadoopPageRank.zip) in this assignment:

- The source code of Hadoop PageRank you implemented.
- Technical report (username_HadoopPageRank_report.docx) that contains:
- The description of the main steps and data flow in your program.
- The output file (username_HadoopPageRank_output.txt) which contains the first 10 urls along with their ranks.

Evaluation

The point total for this project is 10, where the distribution is as follows:

- Completeness of your code and output (7 points)
- Correctness of written report (3 points)

1 What is PageRank?

The web search engine is a typical distributed system on the Internet. It is designed to search for information on the World Wide Web. The search results are generally presented in a list of results and are often called hits. PageRank is a well-known web graph ranking algorithm that helps Internet users sort hits by their importance.

PageRank calculates a numerical value for each element of a hyperlinked set of webpages, which reflects the probability that a random surfer will access that page. The process of PageRank can be understood as a Markov Chain which requires iterative calculations to converge. An iteration of PageRank calculates the new access probability for each webpage based on values calculated in the previous iteration. The process will repeat until the number of current iterations is bigger than predefined maximum iterations, or the Euclidian distance between rank values in two subsequent iterations is less than a predefined threshold that controls the accuracy of the output results.

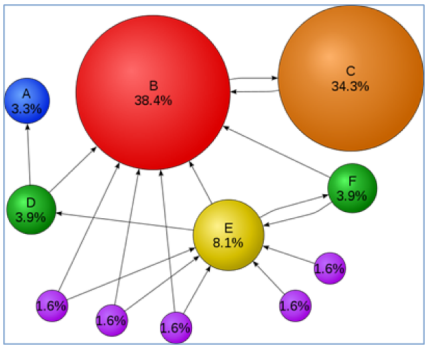


Figure 1: Mathematical PageRank for a simple network in Wikipedia

Figure 1 shows a web graph consisting of 11 vertices A, B, C, D, E, F, G1, G2, G3, G4, G5. Each vertex refers to a unique webpage, and the directed edge means there is one link from the source webpage to the target webpage. The percentage on each vertex represents the rank value of each webpage.

Notes

You can implement a sequential PageRank that can run on desktops or laptops. But when processing larger input data, like web graphs containing more than a million webpages, you need to run the PageRank application in parallel so that it can aggregate the computing power of multiple compute nodes. Currently, in both industry and academia, the study of large-scale web or social graphs has become increasingly popular. In one published paper, the job execution engines that claim to support large-scale PageRank include: MPI, Hadoop, Dryad, Twister, Pregel.

Formula

Equation 1 is the formula to calculate the rank value for each webpage. We will learn this formula by applying it to the case in Figure 1. There are 11 webpages in Figure 1, which include: A, B, C, D, E, F, G1, G2, G3, G4, G5. Assuming the probability distribution for a web surfer accessing all these 11 pages in current iteration is {PR(A), PR(B), PR(C), ... PR(G5)}, then the probability for the surfer to access Page B in the next iteration is:

$$PR(B) = PR(D)/2 + PR(E)/3 + PR(F)/2 + PR(C) + PR(G1)/2 + PR(G2)/2 + PR(G3)/2$$

In a general case, the PageRank value for any page u can be expressed as:

$$PR(u) = \sum_{v \in Set} \frac{PR(V)}{L(v)} \tag{1}$$

The vertices seen in the right of the formula contain all the webpages that point to target webpage 'u'. The L(v) refers to the out degree of each webpage in the vertices set. The initial rank values of each webpage, like PR'(u), can be any double value. After several iteration calculations, the rank values converge to the stationary distribution regardless of what their initial values are.

Damping factor

The PageRank theory holds that even an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue is a damping factor d. Various studies have tested different damping factors, but it is generally assumed that the damping factor will be around 0.85. The formula considering damping factor is shown in Equation 2. N refers to the total number of unique urls.

$$PR(u) = \frac{1-d}{N} + d * \sum_{v \in Set} \frac{PR(V)}{L(v)} \tag{2}$$

Hadoop PageRank DataFlow

In this project, we have provided a sketch code which contains three MapReduce jobs for you to implement:

- CreateGraph (done): add one column, 'initial pagerank value', to the input pagerank adjacency matrix (AM). Then pass it to the PageRank program to calculate the pagerank values.
- PageRank (your implementation): take the transformed AM matrix and calculate pagerank values for all pages.
- Cleanup Results: remove the targetUrls column and output **(sourceUrl, pagerank value)** as the final result.

The detail dataflow can be seen in Figure 2. Part 1 and Part 3 are given as full solutions in this pipeline; you will implement the 2nd part of the PageRank program.

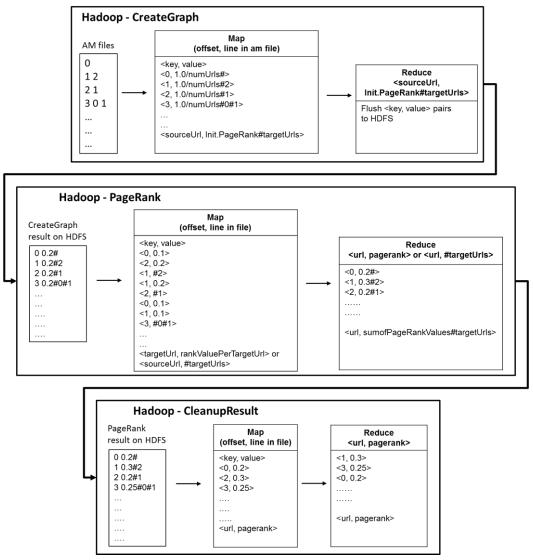


Figure 2: Hadoop PageRank dataflow

Normally for any Hadoop MapReduce program, input data is uploaded and stored in the Hadoop Distributed File System (HDFS) before computation in order to generate **(key, value)** pairs to the mapper. Initially, the PageRank input data is stored in the format of adjacency matrix as a file(s) in the local file system. Then it will be uploaded to the HDFS and distributed across the compute nodes. Hadoop framework reads the application records from HDFS with the InputFormat interface and generates **(key, value)** pair input streams. Each Map function produces zero or more intermediate **(key, value)** pairs by consuming one input (key, value) pair. For this PageRank program, the map function applies the calculation $\frac{PR(v)}{L(V)}$ to each **(key, value)** pair, where the key is the unique id or name of the webpage and the value contains the current rank value of the webpage and its out link information. Map tasks then generate intermediate (key, value) pairs, whose value is the partial rank value of every webpage. Each reduce task aggregates all the partial values of specific webpages by applying the provided Equation 2. The aggregated global rank values are written back to HDFS, which in turn is used as input in the next set of iterations, if any. "Hadoop - PageRank" in Figure 2 shows an example for the PageRank data processing.

Code for Hadoop PageRank

You need to complete two files in the provided pacakge inside "indiana/cgl/hadoop/pagerank/": PageRankMap.java and PageRankReduce.java. Code snapshots are shown below.

```
/*file: PageRankMap.java*/
package indiana.cgl.hadoop.pagerank;

import java.io.BufferedWriter;
/* see detail in source code */

public class PageRankMap extends Mapper<LongWritable, Text, LongWritable, Text> {

  //each map task handles one line within an adjacency matrix file
  // key: file offset
  // value: <sourceUrl PageRank#targetUrls>
  public void map(LongWritable key, Text value, Context context)
  throws IOException, InterruptedException {

    int numUrls = context.getConfiguration().getInt("numUrls",1);
    String line = value.toString();
    StringBuffer sb = new StringBuffer();
    //instance an object that records the information for one webpage
    RankRecord rrd = new RankRecord(line);
    int sourceUrl, targetUrl;
    //double rankValueOfSrcUrl;

    if (rrd.targetUrlsList.size()<=0){
    //there is no out degree for this webpage;
    //scatter its rank value to all other urls
      double rankValuePerUrl = rrd.rankValue/(double)numUrls;
      for (int i=0;i<numUrls;i++){
        context.write(new LongWritable(i), new Text(String.valueOf(rankValuePerUrl)));
      }
    } else {
          /*Write your code here*/
      }     //for
      context.write(new LongWritable(rrd.sourceUrl), new Text(sb.toString()));
  }//map
}
```

```
/*file: PageRankReducer.java*/
package indiana.cgl.hadoop.pagerank;

import java.io.BufferedWriter;
/* see detail in source code */

public class PageRankReduce extends Reducer<LongWritable, Text, LongWritable, Text>{
```

```
  public void reduce(LongWritable key, Iterable<Text> values,
      Context context) throws IOException, InterruptedException {
    double sumOfRankValues = 0.0;
    String targetUrlsList = "";

    int sourceUrl = (int)key.get();
    int numUrls = context.getConfiguration().getInt("numUrls",1);

    //hint: each tuple may include rank value tuple or link relation tuple
    for (Text value: values){
      String[] strArray = value.toString().split("#");
        /*Write your code here*/
    }
              // calculate using the formula
    sumOfRankValues = 0.85*sumOfRankValues+0.15*(1.0)/(double)numUrls;
    context.write(key, new Text(sumOfRankValues+targetUrlsList));
  }
}
```

Edit

The sketch code is stored within the provided VirtualBox image. Use Eclipse or linux text editor vi/vim to add your code.

```
$ cd /root/MoocHomeworks/HadoopPageRank/
$ vim src/indiana/cgl/hadoop/pagerank/PageRankMap.java
$ vim src/indiana/cgl/hadoop/pagerank/PageRankReduce.java
```

Compile and run your code

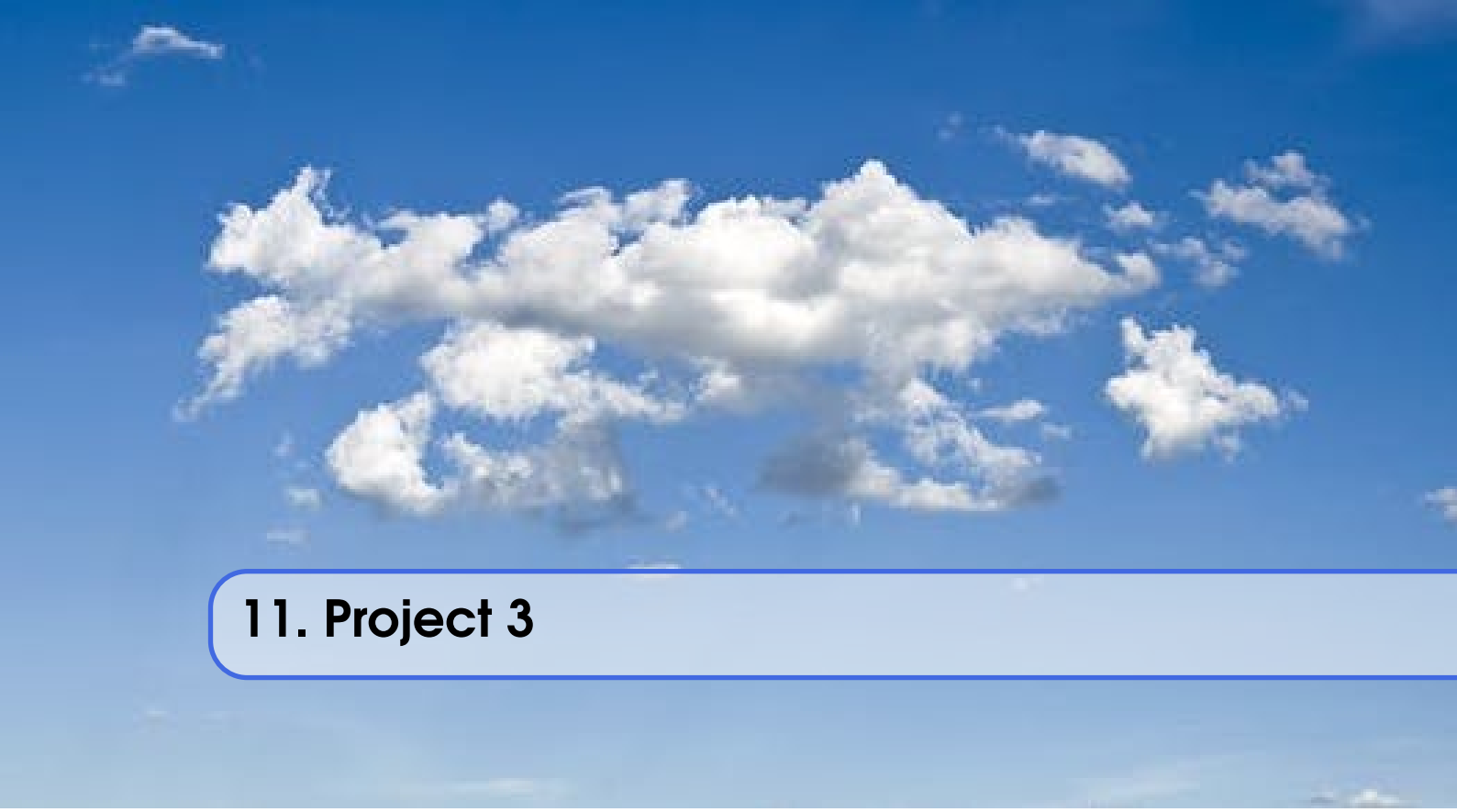
Use the one-click script compileAndExecHadoopPageRank.sh provided below. Standard error messages such as compile errors, execution errors, etc. will be redirected on the screen. Debug them based on the returned messages.

```
$ cd /root/MoocHomeworks/HadoopPageRank/
# usage: ./compileAndExecHadoopPageRank.sh [PageRank Input File][Number of Urls][Number Of
    Iterations]
$ ./compileAndExecHadoopPageRank.sh PageRankDataGenerator/pagerank5000g50.input.0 5000 1
```

View the result

The result is generated as /root/hbaseMoocAntProject/output/project2.txt.

```
$ cd /root/MoocHomeworks/HadoopPageRank/
$ cat output/*
```



11. Project 3

Project 3 asks you to implement a bioinformatics application using Hadoop (Map only) MapReduce framework and write a report about the data flow and your observations of the program.

You are required to turn in the following items in a zip file (username_HadoopBlast.zip) in this assignment:

- The source code of Hadoop Blast you implemented.
- Technical report (username _HadoopBlast_report.docx) that answers the following questions. - What is Hadoop Distributed Cache and how is it used in this program? - Write the two lines that put and get values from Distributed cache. Also include the method and class information. - In previous projects we used Hadoop's TextInputFormat to feed in the file splits line by line to map tasks. In this program, however, we want to feed in a whole file to a single map task. What is the technique used to achieve this? Also, briefly explain what are the key and value pairs you receive as input to a map task and what methods are responsible for producing these pairs? - Do you think this particular implementation will work if the input files are larger than the default HDFS block size? Briefly explain why. [Hint: you can test what will happen by concatenating the same input file multiple times to create a larger input file in the resources/blast_input folder] - If you wanted to extend this program such that all output files will be concatenated into a single file, what key and value pairs would you need to emit from the map task? Also, how would you use these in the reduce that you would need to add?
- The 4 output FASTA files – celllines_1.fa to celllines_4.fa.

Points will be reduced (maximum 0.5 points) if the filename or directory structure are different from instructed above.

The point total for this project is 3, where the distribution is as follows:

- Completeness of your code and output (1 points)
- Correctness of written report (2 points)

Project 3: Hadoop Blast
Cloud Computing
Spring 2017

Professor Judy Qiu

Goal

By this point you should have gone over the sections concerning Hadoop Setup and a few Hadoop programs. Now you are going to blend these applications by implementing a parallel version of BLAST (Basic Local Alignment Search Tool: `http://blast.ncbi.nlm.nih.gov/Blast.cgi`) using the programming interfaces of the Hadoop MapReduce framework. Note that this application is written in "Map-Only" fashion, which means no reduce code is necessary.

Deliverables

You are required to turn in the following items in a zip file (username_HadoopBlast.zip)

- The source code of Hadoop Blast you implemented.
- Technical report (username_HadoopBlast_report.docx) that answers the following questions.
  - What is Hadoop Distributed Cache and how is it used in this program?
  - Write the two lines that put and get values from Distributed cache. Also include the method and class information.
  - In previous projects we used Hadoop's TextInputFormat to feed in the file splits line by line to map tasks. In this program, however, we want to feed in a whole file to a single map task. What is the technique used to achieve this? Also, briefly explain what are the key and value pairs you receive as input to a map task and what methods are responsible for producing these pairs?
  - Do you think this particular implementation will work if the input files are larger than the default HDFS block size? Briefly explain why. [Hint: you can test what will happen by concatenating the same input file multiple times to create a larger input file in the resources/blast_input folder]
  - If you wanted to extend this program such that all output files will be concatenated into a single file, what key and value pairs would you need to emit from the map task? Also, how would you use these in the reduce that you would need to add?
- The 4 output FASTA files: celllines_1.fa to celllines_4.fa.

Evaluation

The point total for this project is 3, where the distribution is as follows:

- Completeness of your code and output (1 points)
- Correctness of written report (2 points)

Introduction

Hadoop-Blast is an advanced Hadoop program which helps BLAST, a bioinformatics application, to utilize the computing capability of Hadoop. This exercise shows the details of its implementation, and provides an example of how to handle similar approaches in other applications.

BLAST is one of the most widely used bioinformatics applications written in C++. The version we are using is v2.2.23, which houses new features and better performance. The database used in the following settings is a subset of a full 8.5GB(nr)database; its full name is Non-redundant protein sequence database. Optionally, for more details on how to run the BLAST binary, please see Big Data for Science tutorial page for Blast Installation [NOT required for the assignment].

In this project, we have provided a sketch code which contains just one java class for you to implement:

- RunnerMap.java: The pleasingly-parallel/map-only Map class which takes the prepackaged Blast (v2.2.23) Binary Program and optimized database from Hadoop's Distributed Cache, then executes BLAST binary as java external process with the assigned FASTA file. These are passed as key-value pairs of **(filename, filepath on HDFS)** handled by a provided customized Hadoop MapReduce InputFormat DataFileInputFormat.java.

The detail dataflow can be seen in Figure 1. You will implement the RunnerMap.java, which copies the distributed cache and assigned FASTA file to local, then run the BLAST binary with correct parameters.

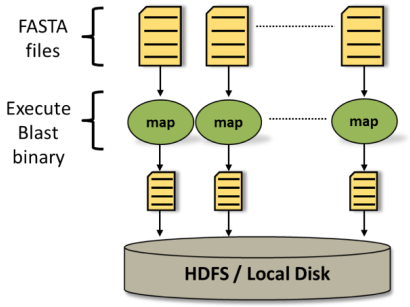


Figure 1: Hadoop Blast dataflow

Normally, for any Hadoop MapReduce program, input data is uploaded and stored in the Hadoop Distributed File System (HDFS) before computation in order to generate **(key, value)** pairs to the mapper. Initially, the BLAST input data is a set of FASTA files located in the local file system. Then it will be uploaded to the HDFS and distributed across the compute nodes. Hadoop framework reads the application records from HDFS with the InputFormat interface and generates **(key, value)** pair input streams; here, we use a provided customized Hadoop MapReduce InputFormat DataFileInputFormat.java to generate key-value pairs of **(filename, filepath on HDFS)**. For this Hadoop Blast program, the map function initially sets up the distributed cache and generates the two absolute location filepaths for Blast binary and Blast Database. Afterwards it copies the assigned FASTA file to local disk by looking up the file from HDFS and generating an absolute filepath. Once this is accomplished and file dependencies are stored in the local disk, we call an external java process and execute the Blast binary with the correct parameters. Finally, the output FASTA file of Blast binary will be uploaded back to HDFS.

Sketch for Hadoop Blast

You need to complete one file in the provided pacakge inside "cgl/hadoop/apps/runner": RunnerMap.java. Code snapshots are shown below.

```
/*file: RunnerMap.java*/
package cgl.hadoop.apps.runner;

import java.io.File;
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

/**
 * @author Thilina Gunarathne (tgunarat@cs.indiana.edu)
 *
 * @editor Stephen, TAK-LON WU (taklwu@indiana.edu)
 */

public class RunnerMap extends Mapper<String, String, IntWritable, Text> {

  private String localDB = "";
  private String localBlastProgram = "";


  @Override
  public void setup(Context context) throws IOException{
    Configuration conf = context.getConfiguration();
    Path[] local = DistributedCache.getLocalCacheArchives(conf);

    /* Write your code here
          get two absolute filepath for localDB and localBlastBinary
        */

  }


  public void map(String key, String value, Context context) throws IOException,
    InterruptedException {

    long startTime = System.currentTimeMillis();
    String endTime = "";

    Configuration conf = context.getConfiguration();
    String programDir = conf.get(DataAnalysis.PROGRAM_DIR);
    String execName = conf.get(DataAnalysis.EXECUTABLE);
    String cmdArgs = conf.get(DataAnalysis.PARAMETERS);
    String outputDir = conf.get(DataAnalysis.OUTPUT_DIR);
    String workingDir = conf.get(DataAnalysis.WORKING_DIR);

      String localInputFile = null;
      String outFile = null;
      String stdOutFile = null;
      String stdErrFile = null;

    System.out.println("the map key : " + key);
    System.out.println("the value path : " + value.toString());
    System.out.println("Local DB : " + this.localDB);

        /*
          Write your code to get localInputFile, outFile,
          stdOutFile and stdErrFile
        */

```

```
68
69      // Prepare the arguments to the executable
70      String execCommand = cmdArgs.replaceAll("#_INPUTFILE_#", localInputFile);
71      if (cmdArgs.indexOf("#_OUTPUTFILE_#") > -1) {
72        execCommand = execCommand.replaceAll("#_OUTPUTFILE_#", outFile);
73      }else{
74        outFile = stdOutFile;
75      }
76
77      endTime = Double.toString(((System.currentTimeMillis() - startTime) / 1000.0));
78      System.out.println("Before running the executable Finished in " + endTime + " seconds");
79
80      execCommand = this.localBlastProgram + File.separator + execName
81      + " " + execCommand + " -db " + this.localDB;
82      //Create the external process
83
84      startTime = System.currentTimeMillis();
85
86      Process p = Runtime.getRuntime().exec(execCommand);
87
88      OutputHandler inputStream = new OutputHandler(p.getInputStream(), "INPUT", stdOutFile);
89      OutputHandler errorStream = new OutputHandler(p.getErrorStream(), "ERROR", stdErrFile);
90
91      // start the stream threads.
92      inputStream.start();
93      errorStream.start();
94
95      p.waitFor();
96      //end time of this procress
97      endTime = Double.toString(((System.currentTimeMillis() - startTime) / 1000.0));
98      System.out.println("Program Finished in " + endTime + " seconds");
99
100     //Upload the results to HDFS
101     startTime = System.currentTimeMillis();
102
103     Path outputDirPath = new Path(outputDir);
104     Path outputFileName = new Path(outputDirPath,fileNameOnly);
105     fs.copyFromLocalFile(new Path(outFile),outputFileName);
106
107     endTime = Double.toString(((System.currentTimeMillis() - startTime) / 1000.0));
108     System.out.println("Upload Result Finished in " + endTime + " seconds");
109
110   }
111 }
```

In addition, if you need to understand the dataflow and main program, please look into the DataAnalysis.java.

```
1  /*file: DataAnalysis.java*/
2  package cgl.hadoop.apps.runner;
3
4  import org.apache.hadoop.conf.Configuration;
5  import org.apache.hadoop.conf.Configured;
6  import org.apache.hadoop.filecache.DistributedCache;
7  import org.apache.hadoop.fs.FileSystem;
8  import org.apache.hadoop.fs.Path;
9  import org.apache.hadoop.io.IntWritable;
10 import org.apache.hadoop.io.Text;
11 import org.apache.hadoop.mapreduce.Job;
12 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14 import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
15 import org.apache.hadoop.util.Tool;
16 import org.apache.hadoop.util.ToolRunner;
17
18 import cgl.hadoop.apps.DataFileInputFormat;
19 import java.net.URI;
20
```

```
21 public class DataAnalysis extends Configured implements Tool {
22
23   public static String WORKING_DIR = "working_dir";
24   public static String OUTPUT_DIR = "out_dir";
25   public static String EXECUTABLE = "exec_name";
26   public static String PROGRAM_DIR = "exec_dir";
27   public static String PARAMETERS = "params";
28   public static String DB_NAME = "nr";
29   public static String DB_ARCHIVE = "BlastDB.tar.gz";
30
31 /**
32  * Launch the MapReduce computation.
33  * This method first, remove any previous working directories and create a new one
34  * Then the data (file names) is copied to this new directory and launch the
35  * MapReduce (map-only though) computation.
36  * @param numMapTasks - Number of map tasks.
37  * @param numReduceTasks - Number of reduce tasks =0.
38  * @param programDir - The directory where the Cap3 program is.
39  * @param execName - Name of the executable.
40  * @param dataDir - Directory where the data is located.
41  * @param outputDir - Output directory to place the output.
42  * @param cmdArgs - These are the command line arguments to the Cap3 program.
43  * @throws Exception - Throws any exception occurs in this program.
44  */
45   void launch(int numReduceTasks, String programDir,
46       String execName, String workingDir, String databaseArchive, String databaseName,
47       String dataDir, String outputDir, String cmdArgs) throws Exception {
48
49     Configuration conf = new Configuration();
50     Job job = new Job(conf, execName);
51
52     // First get the file system handler, delete any previous files, add the
53     // files and write the data to it, then pass its name as a parameter to
54     // job
55     Path hdMainDir = new Path(outputDir);
56     FileSystem fs = FileSystem.get(conf);
57     fs.delete(hdMainDir, true);
58
59     Path hdOutDir = new Path(hdMainDir, "out");
60
61     // Starting the data analysis.
62     Configuration jc = job.getConfiguration();
63
64     jc.set(WORKING_DIR, workingDir);
65     jc.set(EXECUTABLE, execName);
66     jc.set(PROGRAM_DIR, programDir); // this the name of the executable archive
67     jc.set(DB_ARCHIVE, databaseArchive);
68     jc.set(DB_NAME, databaseName);
69     jc.set(PARAMETERS, cmdArgs);
70     jc.set(OUTPUT_DIR, outputDir);
71
72
73     long startTime = System.currentTimeMillis();
74     DistributedCache.addCacheArchive(new URI(programDir), jc);
75     System.out.println("Add Distributed Cache in "
76         + (System.currentTimeMillis() - startTime) / 1000.0
77         + " seconds");
78
79
80     FileInputFormat.setInputPaths(job, dataDir);
81     FileOutputFormat.setOutputPath(job, hdOutDir);
82
83     job.setJarByClass(DataAnalysis.class);
84     job.setMapperClass(RunnerMap.class);
85     job.setOutputKeyClass(IntWritable.class);
86     job.setOutputValueClass(Text.class);
87
```

```
88      job.setInputFormatClass(DataFileInputFormat.class);
89      job.setOutputFormatClass(SequenceFileOutputFormat.class);
90      job.setNumReduceTasks(numReduceTasks);
91
92      startTime = System.currentTimeMillis();
93
94      int exitStatus = job.waitForCompletion(true) ? 0 : 1;
95      System.out.println("Job Finished in "
96          + (System.currentTimeMillis() - startTime) / 1000.0
97          + " seconds");
98      //clean the cache
99
100
101     System.exit(exitStatus);
102   }
103
104   public int run(String[] args) throws Exception {
105     if (args.length < 8) {
106       System.err.println("Usage: DataAnalysis <Executable and Database Archive on HDFS>
107        <Executable> <Working_Dir> <Database dir under archive> <Database name>
108         <HDFS_Input_dir> <HDFS_Output_dir> <Cmd_args>");
109       ToolRunner.printGenericCommandUsage(System.err);
110       return -1;
111     }
112     String programDir = args[0];
113     String execName = args[1];
114     String workingDir = args[2];
115     String databaseArchive = args[3];
116     String databaseName = args[4];
117     String inputDir = args[5];
118     String outputDir = args[6];
119     //"#_INPUTFILE_# -p 95 -o 49 -t 100"
120     String cmdArgs = args[7] ;
121
122     int numReduceTasks = 0;// We don't need reduce here.
123
124     launch(numReduceTasks, programDir, execName, workingDir ,
125       databaseArchive, databaseName,inputDir, outputDir, cmdArgs);
126     return 0;
127   }
128
129   public static void main(String[] argv) throws Exception {
130     int res = ToolRunner.run(new Configuration(), new DataAnalysis(), argv);
131     System.exit(res);
132   }
133 }
```

Edit

The sketch code is stored within the provided VirtualBox image. Use linux text editor vi/vim to add your code.

```
1 $ cd /root/MoocHomeworks/HadoopBlast/
2 $ vim src/cgl/hadoop/apps/runner/RunnerMap.java
```

Compile and run your code

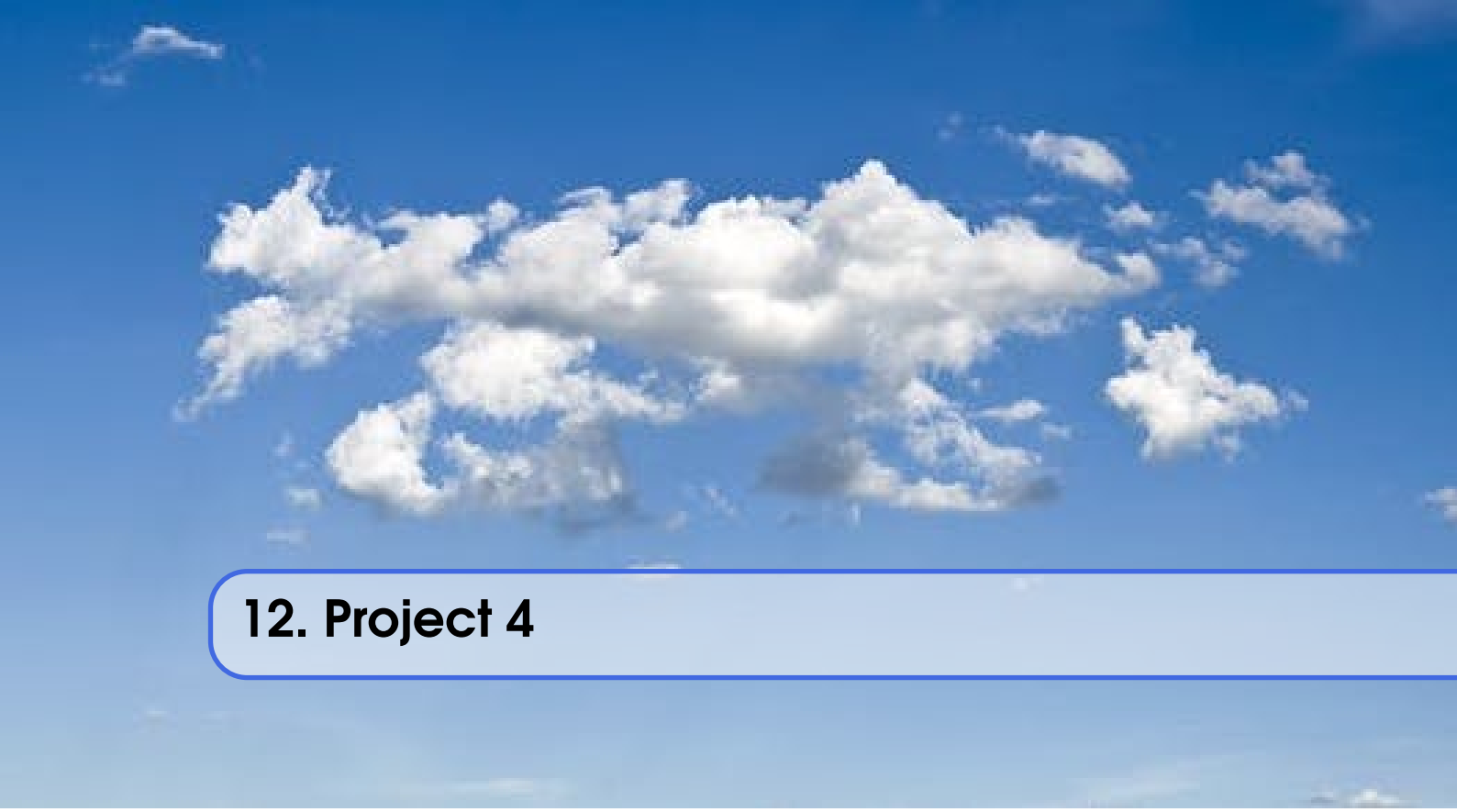
Use the same one-click script compileAndExecHadoopBlast.sh as in prior homework. Standard error messages such as compile errors, execution errors, etc. will be redirected on the screen. Follow the same debugging format.

```
1 $ cd /root/MoocHomeworks/HadoopBlast/
2 $ ./compileAndExecHadoopBlast.sh
```

View the result

The result is generated at /root/MoocHomeworks/HadoopBlast/output/HDFS_blast_output . There should be 4 output FASTA files with .fa extension

```
$ cd /root/MoocHomeworks/HadoopBlast/output/HDFS_blast_output
$ ls
```



12. Project 4

Zip your source code and report in a file named username_project4.zip

The point total for this project is 1.5, where the distribution is as follows:

- Correctness of your code and output (1 points)
- Completeness of written report (0.5 points)

Before you start this project, you need to complete the Project4 Prerequisite<files/project4_pre.pdf> first. The submission folder for it will be published before the lab session.

Project 4: HBase WordCount
Cloud Computing
Spring 2017

Professor Judy Qiu

Goal

Write an HBase WordCount program to count all unique terms' occurrences from the clueWeb09 dataset. Each row record of columnfamily "frequencies" is unique; the rowkey is the unique term stored in byte format, column name is "count" and value is the term frequency shown in all documents. Load the result to HBase WordCountTable. Figure 1 shows the schema of WordCountTable. You will compare the results of your finished run to a correct version we will supply to you.

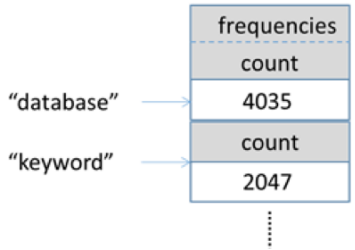


Figure 1: WordCount table schema for storing unique term's occurrences

Deliverables

Zip your source code and report in a file named username_project4.zip

Evaluation

The point total for this project is 1.5, where the distribution is as follows:

- Correctness of your code and output (1 points)
- Completeness of written report (0.5 points)
- The report should explain the logic behind your code.

Prerequisites

You?ll need to load data to HBase before trying this assignment. Please follow **project4_Prereq.pdf** for more information.

Introduction

WordCount is a simple program which counts the number of occurrences of each word in a given text input dataset. It fits very well with the map/reduce programming model, making WordCount a great example to understand the Hadoop MapReduce programming style. Instead of loading the data from HDFS, we will load our data directly from existing HBase records which store the similar content structures on HBase and HDFS.

In this homework and the next homework (Building an Inverted Index) we use the same source code, which can be found in: */root/MoocHomeworks/HBaseWordCount*.

Clueweb09 dataset

We are using the ClueWeb09 dataset, which was created to support research on information retrieval and related human language technologies. It consists of about 1 billion webpages in ten languages that were collected in January and February 2009. The dataset is used by several tracks of the TREC conference[2]. Since the ClueWeb09 dataset is composed of webpages crawled from the Internet, the uploaded table schemas are designed as shown in Figure 2.

Figure 2: Data table schema for storing the ClueWeb09 dataset

So, while similar to Hadoop WordCount [3], the differences are that data is stored on HBase and URI is the "filename" that contains all the text content.

Mapper, Reducer and Main Program

Now we are going to implement the HBase WordCount. Our implementation consists of three main parts:

- Mapper
- Reducer
- Main program

Mapper

A Mapper overrides the map function from the Class "org.apache.hadoop.hbase.mapreduce.TableMapper<Text, LongWritable>" which provides <key, value> pairs as the input. A Mapper implementation may output <key, value> pairs using the provided Context. <key, value> of this map function is <rowkey, content>, where the key is the rowkey of an HBase record related to a specified URI, and the content is the stored text of that URI. Your Map task should output <word, frequency> for each word in the content of text.

Pseudocode

```
void Map (key, value){
    for each word x in the content of a hbase record:
    context.write(x, freq);
}
```

Detailed implementation

```
static class WcMapper extends TableMapper<Text, LongWritable> {
    @Override
    public void map(ImmutableBytesWritable row, Result result, Context context) throws
    IOException, InterruptedException {
      byte[] contentBytes = result.getValue(Constants.CF_DETAILS_BYTES, Constants.
    QUAL_CONTENT_BYTES);
      String content = Bytes.toString(contentBytes);

      // TODO: write your implementation for counting words in each row, and generating a <
    word, count> pair
      // Hint: use the "getWordFreq" function to count the frequencies of words in content

    }
}
```

Reducer

A Reducer collects the intermediate <key, value> output from multiple map tasks and assembles a single result. Here, the reducer function will sum up the occurrence of each word to pairs as <word, occurrence >, then write it back to an HBase table with put operations which contain the key-value pair information of each word. *Pseudocode*

```
void Reduce (keyword, <list of value>){
    for each x in <list of value>:
        sum+=x;
        context.write(rowkey(x), freq);
}
```

Detailed implementation

```
public static class WcReducer extends TableReducer<Text, LongWritable,
    ImmutableBytesWritable> {
      @Override
    public void reduce(Text word, Iterable<LongWritable> freqs, Context context)
                throws IOException, InterruptedException {
        /*TODO: write your implementation for getting the final count of each word
        and putting it into the word count table
        Hint -- the schema of the WordCountTable is:
           rowkey: a word, column family: "frequencies",
           column name: "count", cell value: count of the word
        Check iu.pti.hbaseapp.Constants for the constant values to use.
  */
      long totalFreq = 0;
     }
}

```

Main program

The main function has been provided as standard initialization, although you can modify it to suit your own style. Hint: the provided code is designed for using put operations in the reducer content.write() function. Before writing the codes, please read the HBase MapReduce tutorial first [4].

Edit

The sketch code is stored within the provided VirtualBox image Environment Setup. You may use linux text editor vi/vim to add your code.

```
$ cd /root/MoocHomeworks/HBaseWordCount/
$ vim src/iu/pti/hbaseapp/clueweb09/WordCountClueWeb09.java

```

Compile and run your code

For your convenience, we have provided a one-click script compileAndExecWordCount.sh for compiling and execution. Standard error messages such as "compile errors, execution errors, etc." will be redirected on the screen. You may debug it based on the returned messages.

```
$ cd /root/MoocHomeworks/HBaseWordCount
$ ./compileAndExecWordCount.sh
```

View the result

The result is generated as /root/MoocHomeworks/HBaseWordCount/output/project1.txt.

```
$ cd /root/MoocHomeworks/HBaseWordCount
$ cat output/project1.txt
```

References

[1] Clueweb09 dataset. `http://lemurproject.org/clueweb09/`.

[2] Hadoop WordCount. `http://salsahpc.indiana.edu/csci-b649-spring-2014/projects/project1.html`.

[3] HBase MapReduce Examples. `http://hbase.apache.org/book/mapreduce.example.html`.

Load Data into HBase

Goal

This is a warm-up practice to provide background knowledge of Hadoop MapReduce and HBase Database/Datastore. The following practice will cover:

- Overview for Hadoop and HBase
- Set up the environment for Hadoop and HBase
- Create HBase Database tables for ClueWeb09 [1] data, and load them into HBase
- Test the stored HBase records with the provided java executable

Deliverables

Submit an output file dataTable1.txt containing your results to the Canvas Assignments page by using the provided java class iu.pti.hbaseapp.HBaseTableReader.

Evaluation

None

Introduction

The development of data-intensive problems in recent years has brought new requirements and challenges to storage and computing infrastructures. Researchers are not only doing batch loading and processing of large-scale data, but also demanding the capabilities of incremental updating and interactive analysis. Therefore, extending existing storage systems to handle these new requirements becomes an important research challenge.
In this exercise, we will introduce technologies related to data-intensive computing and storage, namely Hadoop and HBase.

Hadoop MapReduce

The Apache Hadoop MapReduce software library is a framework that allows for the distributed processing of large datasets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high availability, the library itself is designed to detect and handle failures at the application layer, delivering a highly available service on top of a cluster of computers, each of which may be prone to failures [2].
Hadoop has an architecture consisting of a master node with many client workers and uses a global queue for task scheduling, thus achieving natural load balancing among the tasks. The MapReduce

model reduces the data transfer overhead by overlapping data communication with computations when reduce steps are involved. Hadoop performs duplicate executions of slower tasks and handles failures by rerunning the failed tasks using different workers. Data is stored on the Hadoop Distributed File System (HDFS), a distributed parallel file system for data storage, which stores the data across the local disks of the computing nodes while presenting a single file system view through the HDFS API. The architecture is shown in Figure 1.

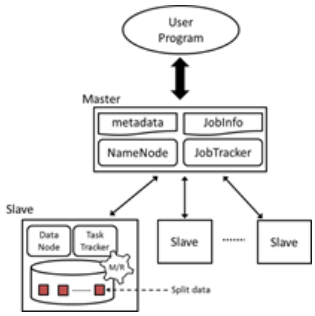


Figure 1. Hadoop MapReduce Architecture

This practice session and upcoming projects mainly use Hadoop MapReduce as our programming model and execution framework.

HBase

HBase is an open source, distributed, column-oriented, and sorted-map datastore modeled after Google's BigTable. Figure 2 shows the data model of HBase. Data is stored in tables; each table contains multiple rows and a fixed number of column families. For every row, there can be a varied amount of qualifiers within a column family, and at the intersections of rows and qualifiers are table cells. Cell contents are both uninterpreted and versioned arrays of bytes. A table can be configured to maintain a certain number of versions for its cell contents. Rows are sorted by row keys, which are implemented as byte arrays [3-4].

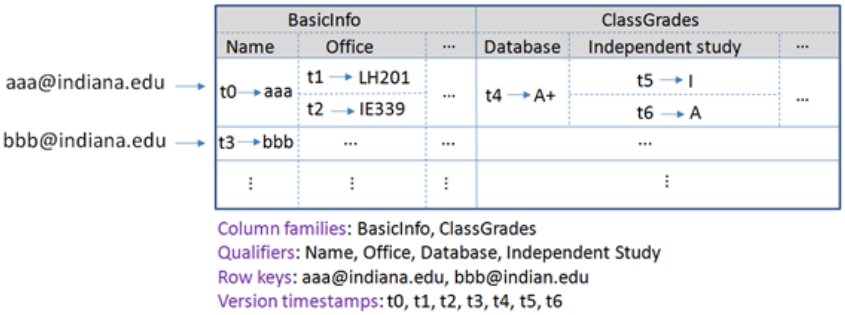


Figure 2. An example of HBase table structure

This practice and upcoming projects use HBase as a datastore which stores the raw input data and the output results, even serving as a database for our search engine.

Exercise: Create Tables and Load Data to HBase

This section provides command-line steps to configure the working environment, start Hadoop and HBase, create HBase tables, load data into HBase, and view the uploaded records. Note that these steps must be executed under the prepared virtual box in your machine.

Start Hadoop and HBase

The following steps direct you to the locations of Hadoop and HBase and give instructions on how to start them. For Hadoop, a provided one-click shell script, MultiNodesOneClickStartUp.sh, is used to automatically start the Hadoop framework. For HBase, we use the default starter.

```
# start hadoop
$ cd /root/software/hadoop-1.1.2/
$ . ./MultiNodesOneClickStartUp.sh /root/software/jdk1.6.0_33/ nodes

# start hbase
$ cd /root/software/hbase-0.94.7/
$ ./bin/start-hbase.sh
```

Configure the working environment

Once Hadoop and HBase have started, we need to copy a configuration file hbase-site.xml to Hadoop's conf directory and update the environment parameter HADOOP_CLASSPATH.

```
# prepare for hadoop and hbase environment

$ cp /root/software/hbase-0.94.7/conf/hbase-site.xml /root/software/hadoop-1.1.2/conf/
$ cd /root/software/hadoop-1.1.2/
$ export HADOOP_CLASSPATH=`/root/software/hbase-0.94.7/bin/hbase classpath`
```

Load data into HBase Tables

After the working environment is ready, two HBase Tables, WordCountTable and clueWeb09DataTable for this homework, are created by the provided java class iu.pti.hbaseapp.clueweb09.TableCreatorClueWeb09. Then we upload the clueWeb09 data to HBase using helper iu.pti.hbaseapp.clueweb09.DataLoaderClueWeb09.

```
export HADOOP_CLASSPATH=`/root/software/hbase-0.94.7/bin/hbase classpath`

# create hbase tables
$ ./bin/hadoop jar lib/cglHBaseMooc.jar iu.pti.hbaseapp.clueweb09.TableCreatorClueWeb09

# create one directory for mapreduce data input
$ mkdir -p /root/MoocHomeworks/HBaseWordCount/data/clueweb09/mrInput

# create input's metadata for HBbase data loader
$ ./bin/hadoop jar lib/cglHBaseMooc.jar iu.pti.hbaseapp.clueweb09.Helpers create-mr-inpu
t /root/MoocHomeworks/HBaseWordCount/data/clueweb09/files/ /root/MoocHomeworks/HBaseWord
Count/data/clueweb09/mrInput/ 1

# copy metadata to Hadoop HDFS
$ ./bin/hadoop dfs -copyFromLocal /root/MoocHomeworks/HBaseWordCount/data/clueweb09/mrIn
put/ /cw09LoadInput
$ ./bin/hadoop dfs -ls /cw09LoadInput

# load data into HBase (takes 10-20 minutes to finish)
$ ./bin/hadoop jar lib/cglHBaseMooc.jar iu.pti.hbaseapp.clueweb09.DataLoaderClueWeb09 /c
w09LoadInput
```

Verify data record from HBase output

Finally, we verify the uploaded records with the iu.pti.hbaseapp.clueweb09.DataLoaderClueWeb09 from the clueWeb09DataTable. The screen redirects the output to dataTable1.txt. This file is used for evaluation and must be uploaded to the Oncourse Assignments page.

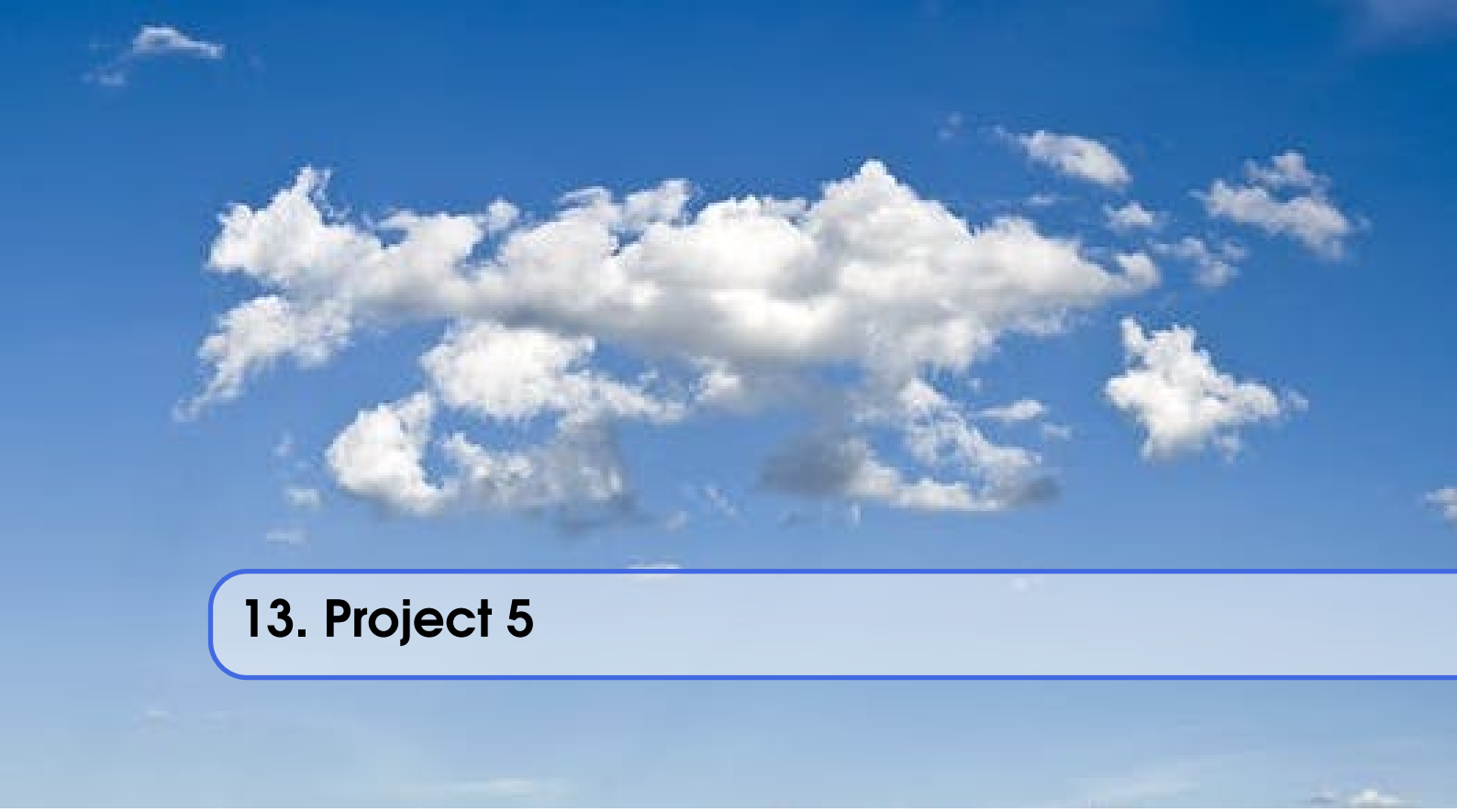
```
  $ ./bin/hadoop jar lib/cglHBaseMooc.jar iu.pti.hbaseapp.HBaseTableReader  clueWeb09Dat
aTable details string string string string 1 > dataTable1.txt

  $ vim dataTable1.txt
```

What's next?

You are ready to program your own HBase WordCount.

References

1. Clueweb09 project, http://lemurproject.org/clueweb09/ (http://lemurproject.org/clueweb09/)
2. Hadoop official website, http://hadoop.apache.org/ (http://hadoop.apache.org/)
3. HBase official website, http://hbase.apache.org/ (http://hbase.apache.org/)
4. Xiaoming Gao, Vaibhav Nachankar, and Judy Qiu. 2011. Experimenting lucene index on HBase in



13. Project 5

Write an HBase FreqIndexBuilder program to build an inverted index table which has the unique term's occurrences in all documents from the clueWeb09 dataset. Zip your source code, results and report in a file named username_project5.zip. Submit this file to the Canvas submission page.

- Complete source code
- A written report describing the main steps

The point total for this project is 3, where the distribution is as follows:

- Completeness of your code and output (2 points)
- Correctness of written report (1 points)

Project 5: HBase FreqIndexBuilder
Cloud Computing
Spring 2017

Professor Judy Qiu

Goal

Write an HBase FreqIndexBuilder program to build an inverted index table which has the unique term's occurrences in all documents from the clueWeb09 dataset. Each row record of columnfamily "frequencies" is unique, where the rowkey is the unique term stored in byte format, column name is the documentId that contains this term, and value is the term frequency shown per document. Note that each row has multiple columns. The result must be loaded to HBase clueWeb09IndexTable. Figure 1 shows the schema of clueWeb09IndexTable.

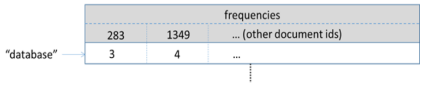


Figure 1: clueWeb09IndexTable table schema for storing term frequencies and their related documentId

Deliverables

Zip your source code, results and report in a file named username_project5.zip. Submit this file to the Canvas submission page.

- Complete source code
- A written report describing the main steps

Evaluation

The point total for this project is 3, where the distribution is as follows:

- Completeness of your code and output (2 points)
- Correctness of written report (1 points)

Introduction

HBase FreqIndexBuilder is an advanced WordCount program which counts the number of occurrences of each word in a given text input dataset and also stores the related document name (identification number) as HBase inverted index records. These Inverted indicesfor text data are built for supporting efficient searches in a huge set of text data.

What is Inverted Index?

Figure 2 shows an example of an inverted index. For a given set of documents, each composed of a series of terms (words), it records the following information: for each term, which subset of documents contains it in their texts.

To build these inverted indices, we reuse the ClueWeb09 dataset from before, which was created to support research on information retrieval and related human language technologies. The dataset is used by several tracks of the TREC conference. New inverted index table schemas are designed as shown in Figure 1.

In the clueWeb09IndexTable table each term will have the same structure, with term as rowkey, values contained in documentId, and the occurrence of the term within this document shown. Our goal is to write an HBase program which generates an inverted index table by extracting the information from the ClueWeb09 dataset.

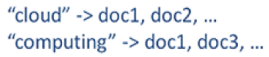


Figure 2: A sample inverted index

Mapper and Main Program

Now we are going to implement the HBase FreqIndexBuilder program. As opposed to WordCount, our implementation only consists of two main parts:

- Mapper
- Main program

This type of application is called ?Map-Only? parallel application.

Mapper

A Mapper overrides the “map” function from the Class “org.apache.hadoop.hbase.mapreduce.TableMapper <Text, LongWritable>”, which provides <key, value>pairs as the input. A Mapper implementation may output <key,value>pairs using the provided Context. <key, value>of this map function is <rowkey, content>, where the key is the rowkey of an HBase record related to a specified URI, and the content is the stored text of that URI. Your Map task should output <word, <docId, frequency>>for each word in the content of text.

Pseudocode

```
void Map(key, value) {
    for each word x in the content of a hbase record:
    context.write(x, );
}
```

Detailed implementation

```
public static class FibMapper extends TableMapper<ImmutableBytesWritable, Writable> {
  @Override
  protected void map(ImmutableBytesWritable rowKey, Result result, Context context) throws
    IOException, InterruptedException {
    byte[] docIdBytes = rowKey.get();
    byte[] contentBytes = result.getValue(Constants.CF_DETAILS_BYTES, Constants.
    QUAL_CONTENT_BYTES);
    String content = Bytes.toString(contentBytes);
```

```
7
8       // TODO: write your implementation for getting the term frequencies from each document,
9       // and generating Put objects for clueWeb09IndexTable.
10              // Hint: use the "getTermFreqs" function to count the frequencies of terms in
        content.
11              // The schema of the clueWeb09IndexTable is:
12              // row key: term, column family: "frequencies", qualifier: document Id, cell
        value: term frequency in the corresponding document
13              // Check iu.pti.hbaseapp.Constants for useful constant values.
14
15
16    }
17  }
```

Main Program

Again, the main function has been provided as standard initialization, and you may modify it to fit your own style. See the examples of using TableMapReduceUtil.initTableMapperJob and TableMapReduceUtil.initTableReducerJob.

Edit, compile and run your code

The sketch code is stored within the provided VirtualBox image. You can use linux text editor vi/vim to add your code.

```
1 $ cd /root/MoocHomeworks/HBaseInvertedIndexing/
2 $ vim src/iu/pti/hbaseapp/clueweb09/FreqIndexBuilderClueWeb09.java
3 $ cd /root/MoocHomeworks/HBaseInvertedIndexing/
4 $ ./compileAndExecFreqIndexBuilderClueWeb.sh
```

View the result

The result is generated as /root/hbaseMoocAntProject/output/project2.txt.

```
1 $ cd /root/MoocHomeworks/HBaseInvertedIndexing/
2 $ cat output/project2.txt
```

14. Project 6

After having familiarized yourself with the “HBase Building an Inverted Index” homework and “PageRank algorithms” homework, you are ready to use these applications to test the search engine function from the packaged executable.

14.1 Deliverables

Zip your source code, library, and results in a file named `username@test-search-engine.zip`. Please submit this file to the Canvas Assignments page.

14.2 Evaluation

The point total for this project is 6, where the distribution is as follows:

- Completeness of your code (5 points)
- Correct output (1 points)

Project 6: Test Search Engine
Cloud Computing
Spring 2017

Professor Judy Qiu

Goal

After having familiarized yourself with the "HBase Building an Inverted Index" homework and "PageRank algorithms" homework, you are ready to use these applications to test the search engine function from the packaged executable.

Deliverables

Zip your source code, library, and results in a file named username@test-search-engine.zip. Please submit this file to the Canvas Assignments page.

Evaluation

The point total for this project is 6, where the distribution is as follows:

- Completeness of your code (5 points)
- Correct output (1 points)

Search Engine Implementation

Before we test the search engine, we need to write the PageRank output to the HBase clueWeb09PageRankTable.

```
$ export HADOOP_CLASSPATH='/root/software/hbase-0.94.7/bin/hbase classpath'
$ hadoop jar /root/software/hadoop-1.1.2/lib/cglHBaseMooc.jar iu.pti.hbaseapp.clueweb09.
    PageRankTableLoader /root/MoocHomeworks/HBaseInvertedIndexing/resources/en0000-01and02.
    docToNodeIdx.txt /root/MoocHomeworks/HBaseInvertedIndexing/resources/en0000-01
    and02_reset_idx_and_square_pagerank.out
```

Now, combined with "Building an Inverted Index", we have built three database tables on HBase:

- clueWeb09DataTable
- clueWeb09IndexTable
- clueWeb09PageRankTable

The data-flow of the program is shown in Figure 1.
You need to complete the following code before you can run the search engine:

```
$ vim src/iu/pti/hbaseapp/clueweb09/SearchEngineTester.java
```

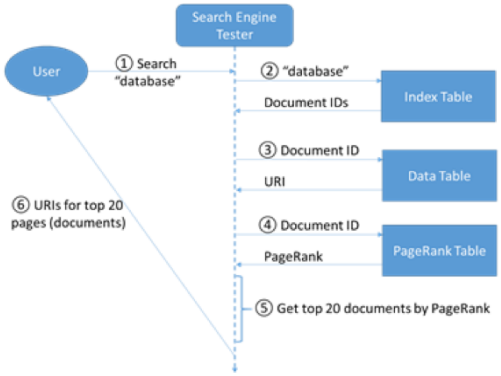


Figure 1: Dataflow for searching keyword “database” among the constructed databases

```
public static void searchKeyword(String keyword) throws Exception {
  Configuration hbaseConfig = HBaseConfiguration.create();
  HTable dataTable = new HTable(hbaseConfig, Constants.CW09_DATA_TABLE_BYTES);
  HTable indexTable = new HTable(hbaseConfig, Constants.CW09_INDEX_TABLE_BYTES);
  HTable prTable = new HTable(hbaseConfig, Constants.CW09_PAGERANK_TABLE_BYTES);

  int topCount = 20;
  // this is the heap for storing the top 20 ranked pages
  PriorityQueue<PageRecord> topPages = new PriorityQueue<PageRecord>(topCount);

  // get the inverted index row with the given keyword
  keyword = keyword.toLowerCase();
    byte[] keywordBytes = Bytes.toBytes(keyword);
  Get gIndex = new Get(keywordBytes);
    Result indexRow = indexTable.get(gIndex);

    // loop through the document IDs in the row. Recall the schema of the
    clueWeb09IndexTable:
    // row key: term (keyword), column family: "frequencies", qualifier: document ID, cell
    value: term frequency in the corresponding document
  int pageCount = 0;
    for (KeyValue kv : indexRow.list()) {
        String pageDocId = null;
        int freq = 0;
        String pageUri = null;
        float pageRank = 0;

      // Write your codes for the main part of implementation here
        // Step 1: get the document ID of one page, as well as the keyword's frequency in
    that page
    // Step 2: get the URI of the page from clueWeb09DataTable
    // Step 3: get the page rank value of this page from clueWeb09PageRankTable
      // End of your code

        // Use the heap to select the top 20 pages according to page rank
      PageRecord page = new PageRecord(pageDocId, pageUri, pageRank, freq);
      if (topPages.size() < topCount) {
      topPages.offer(page);
      }
      else {
      PageRecord head = topPages.peek();
      if (page.pageRank > head.pageRank) {
        topPages.poll();
        topPages.offer(page);
      }
```

```
43        }
44
45        pageCount++;
46        if (pageCount % 100 == 0) {
47        System.out.println("Evaluated " + pageCount + " pages.");
48        }
49    }
50      System.out.println("Evaluated " + pageCount + " pages.");
51    dataTable.close();
52    indexTable.close();
53    prTable.close();
54
55    System.out.println("Evaluated " + pageCount + " pages in total. Here are the top 20 pages
        according to page ranks:");
56    Stack<PageRecord> stack = new Stack<PageRecord>();
57    while (topPages.size() > 0) {
58      stack.push(topPages.poll());
59    }
60    while (stack.size() > 0) {
61      PageRecord page = stack.pop();
62      System.out.println("Document ID: " + page.docId + ", URI: " + page.URI + ", page rank: "
         + page.pageRank + ", word frequency: "
63        + page.termFreq);
64    }
65  }
```

Compile and Run the Program

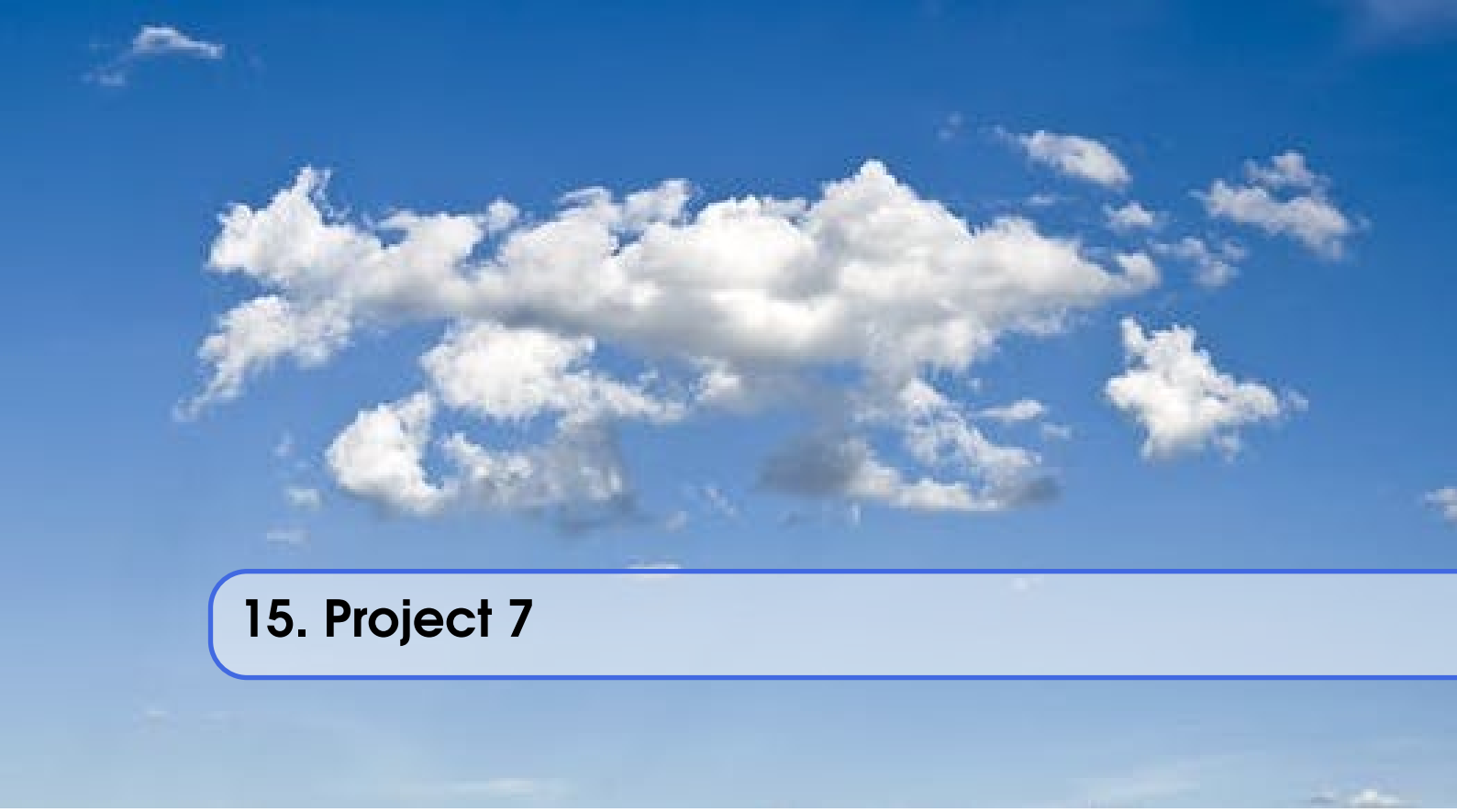
```
1 $ cd /root/MoocHomeworks/HBaseInvertedIndexing/
2 $ vim src/iu/pti/hbaseapp/clueweb09/SearchEngineTester.java
3 $ cd /root/MoocHomeworks/HBaseInvertedIndexing/
4 $ ant
5 $ cp /root/MoocHomeworks/HBaseInvertedIndexing/dist/lib/cglHBaseMooc.jar /root/software/
    hadoop-1.1.2/lib/
```

Now you can test the functionality of the search engine by running the program with keywords.

```
1 $ cd /root/software/hadoop-1.1.2/
2 $ ./bin/hadoop jar lib/cglHBaseMooc.jar  iu.pti.hbaseapp.clueweb09.SearchEngineTester search
    -keyword snapshot
3 $ ./bin/hadoop jar lib/cglHBaseMooc.jar  iu.pti.hbaseapp.clueweb09.SearchEngineTester get-
    page-snapshot 00000113548 |  grep snapshot
```

What's next?

Congratulations, you have finished the search engine project!



15. Project 7

The goal of this project is to familiarize yourself with the concept of map-collective applications. Harp is similar to MapReduce in terms of programming with the exception that it provides collective communication support across map tasks.

Zip your source code and output as username_harp-pagerank.zip. Please submit this file to the Assignments page.

The point total for this project is 6, where the distribution is as follows:

- Completeness of your code (5 points)
- Correct output (1 point)

We prepared a new VM for project7 and project8. Please download it from here.

- VirtualBox VM Download

Project 7: Harp PageRank
Cloud Computing
Spring 2017

Professor Judy Qiu

Goal

For this project you will implement PageRank on Harp[1] framework.

Deliverables

Zip your source code and output as username_harp-pagerank.zip. Please submit this file to the Canvas Assignments page.

Evaluation

The point total for this project is 6, where the distribution is as follows:

- Completeness of your code (5 points)
- Correct output (1 point)

Prerequisites

From now on, you don't need the old VM. To avoid any conflict and inconvenience, we have prepared a new VM (ubuntu 16.04) for you. The link will be posted on canvas. All necessary tools/libaries such Maven, JDK, Github, Hadoop 2.6.0, Harp are configured and ready for use. Intellij is installed as well. You can also use your own VM. But you will need to setup those tools/libraries by yourself. Some tutorials are available at the harp website[1]. Here this instruction is based on the configurations in this new VM.

HarpPageRank Implementation

Most of the code is completed for you and your task will be to perform the **Compute PR** step in the above diagram. The code for this can be found in **simplepagerank/PageRankMapper.java**

```
public void computePartialPR(Map<Long, ArrayList<Long>> partialGraph, Long2DoubleKVTable
    localPRTable, Long2DoubleKVTable globalPRTable){

        for (Entry<Long, ArrayList<Long>> entry : partialGraph.entrySet()) {
            Long sourceUrl = entry.getKey();
            ArrayList<Long> targetUrls = entry.getValue();
            System.out.println("sourceURL: "+sourceUrl);
            if(targetUrls == null ){
                // simply assume that the IDs of pages are: 0,1,2,...,(numUrls-1)
                System.out.println("targetUrls is null");
                double pr = localPRTable.getVal(sourceUrl) / numUrls;
                // TODO - Students write Code here
```

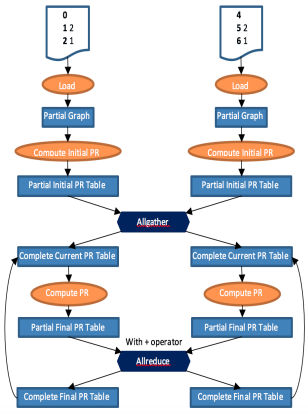


Figure 1: Harp PageRank Architecture

```
                // Add pr to the page rank of all other URLs in globalPRTable
                // Note. The addKeyVal(key, val) method in globalPRTable will
                // automatically sum values if the key exists. If the key does
                // NOT exist then a new entry will be made.


            }else{
                int numOfOutLinks = targetUrls.size();
                double pr = localPRTable.getVal(sourceUrl) / numOfOutLinks;
                // TODO - Students write Code here
                // Add pr to the page rank of all target URLs.



            }
        }
    }
```

Compilation and Running

- To make the modification to the code, you can use Intellij IDE or linux terminal. The source code is located at

```
/home/cc/Documents/harp/harp-tutorial-app/src/main/java/edu/iu/simplepagerank
```

- To compile the code, type the following commands in terminal

```
$ cd $HARP_ROOT_DIR
$ mvn clean package
```

- If hadoop is not started, start hadoop by:

```
$ $HADOOP_HOME/sbin/start-dfs.sh
$ $HADOOP_HOME/sbin/start-yarn.sh
```

  Then you can view the web UI at localhost:50070 and localhost:8088

- We prepared the input dataset (input5K-2partitions) for you. Use the following command to put it to hdfs. Please note there is a "dot" at the end of the second command.

```
$ cd $HARP_ROOT_DIR/data/tutorial/simplepagerank
$ hdfs dfs -put input5K-2partitions .
```

- Run the program:

```
$ cd $HARP_ROOT_DIR
$ hadoop jar harp-tutorial-app/target/harp-tutorial-app-1.0-SNAPSHOT.jar edu.iu.
    simplepagerank.HarpPageRank input5K-2partitions output5k 5000 10
```

  This will run PageRank against input2K-2partitions dataset. It has 5000 URLs in total. The program will run 2 parallel map tasks for 10 iterations. If you want to launch N map tasks, you need to divide the dataset into N partitions.

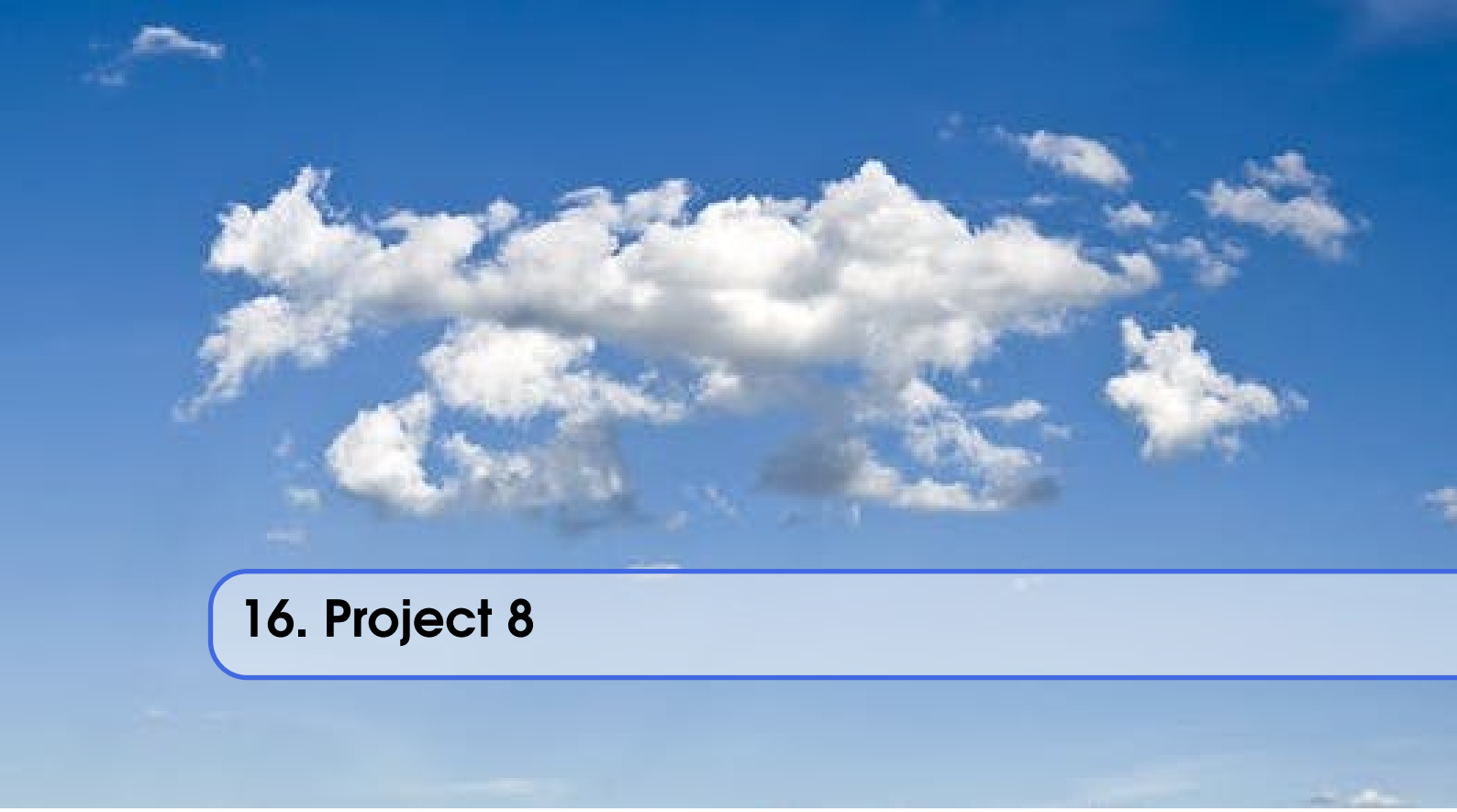
- To get the output, perform the following commands to get the output to the Desktop. Then you can submit it to canvas.

```
$ hdfs dfs -get output5k /home/cc/Desktop
```

References

[1] Indiana University. `https://dsc-spidal.github.io/harp`.

Do not copy and paste commands from pdf files. Please type them manually. Special characters cause problems in executing commands in a terminal.



16. Project 8

Zip your source code and report as username_mbkmeans.zip.

The point total for this project is 6, where the distribution is as follows: - Completeness of your code (5 points) - In the report, describe your implementation and the output. (1 points)

You can get up to 4 bonus points based on your extra efforts.

Project 8: Harp MiniBatch Kmeans
Cloud Computing
Spring 2017

Professor Judy Qiu

Goal

The goal for this project is to implement Harp[1] Mini-batch Kmeans from scratch.

Deliverables

Zip your source code and report as username_mbkmeans.zip. Please submit this file to the Canvas Assignments page.

Evaluation

The point total for this project is 6, where the distribution is as follows:

- Completeness of your code (5 points)
- In the report, describe your implementation and the output. (1 points)

You can get up to 4 bonus points based on your extra efforts.

Bonus credits

Some options you may consider to get extra credits:

- Perform experiments on various (small, medium, large, etc) datasets
- Test your algorithm on at least 2 nodes on FutureSystem.
- Implement mini-batch kmeans using other tools/platforms (Spark[2], Flink[3], etc) and compare the performance between different tools/platforms.

You are encouraged to explore other options to get extra credits. Remember to present all your extra work in the report.

Dataset

You can implement a script to generate data randomly as your input datasets. You are also free to use public datasets such as RCV1-v2[4].

Mini-batch Kmeans

You can refer to the paper[5] for sequential mini-batch kmeans algorithm. You will need to design how to parallelize the algorithm so that it can run with large scale datasets on distribute computing environment.

Algorithm 1 Mini-batch k-Means.

Given: k, mini-batch size b, iterations t, data set X
Initialize each $\mathbf{c} \in C$ with an $\mathbf{x}$ picked randomly from X
$\mathbf{v} \leftarrow 0$
for $i = 1$ to t **do**
  $M \leftarrow b$ examples picked randomly from X
  for $\mathbf{x} \in M$ **do**
    $\mathbf{d}[\mathbf{x}] \leftarrow f(C, \mathbf{x})$ // Cache the center nearest to $\mathbf{x}$
  end for
  for $\mathbf{x} \in M$ **do**
    $\mathbf{c} \leftarrow \mathbf{d}[\mathbf{x}]$ // Get cached center for this $\mathbf{x}$
    $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$ // Update per-center counts
    $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$ // Get per-center learning rate
    $\mathbf{c} \leftarrow (1 - \eta)\mathbf{c} + \eta\mathbf{x}$ // Take gradient step
  end for
end for

Figure 1: Mini-batch Kmeans.[5]

References

[1] Indiana University. `https://dsc-spidal.github.io/harp`.

[2] Apache. `http://spark.apache.org`.

[3] Apache. `https://flink.apache.org`.

[4] David D. Lewis. `http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm`.

[5] David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM, 2010.