Large-Scale Graph Processing Using Apache Giraph

Sherif Sakr · Faisal Moeen Orakzai
Ibrahim Abdelaziz · Zuhair Khayyat

Large-Scale Graph Processing Using Apache Giraph

Springer

Sherif Sakr
School of Computer Science
and Engineering
University of New South Wales
Sydney, NSW
Australia

Ibrahim Abdelaziz
King Abdullah University of Science
and Technology
Thuwal
Saudi Arabia

Faisal Moeen Orakzai
Department of Computer Science
Aalborg University
Aalborg
Denmark

Zuhair Khayyat
King Abdullah University of Science
and Technology
Thuwal
Saudi Arabia

ISBN 978-3-319-47430-4 ISBN 978-3-319-47431-1 (eBook)
DOI 10.1007/978-3-319-47431-1

Library of Congress Control Number: 2016954912

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To my wife, Radwa, my daughter, Jana, and my son, Shehab for their love, encouragement, and support.

Sherif Sakr

To my father, Moeen Orakzai, my mother, Hikmat Moeen, my wife, Sana and my son Ibrahim for their support.

Faisal Moeen Orakzai

To my wife and my lovely daughters for their unconditional love and support.

Ibrahim Abdelaziz

To my father, Yarub, my mother, Nadia, my lovely wife, Manal, and my two handsome boys, Mazin and Ziyad, for their love and support.

Zuhair Khayyat

Foreword

The present decade has been dubbed the Digital Universe Decade because the digital universe, all data available in digital form, is growing at an exponential rate during this decade. Lots of this data is graph data. Examples include data related to social network connections and data diffusion, computer networks, telecommunication networks, the Web, and knowledge bases. Yet another example close to my heart is road-network data: in step with the rapid increases is available vehicle trajectory data, this data grows rapidly in size, resolution, and sophistication.

Graph analytics enables the extraction of valuable information from graph data and enables valuable services on top of graph data. For example, in social media, graph analytics can be used to detect communication patterns that might be of interest to national defense. In the medical and biological domains, graph analytics may be used for analyzing relationships in the contexts of proteins, DNA, cells, chemical pathways, and organs in order to determine how they are affected by lifestyle choices and medications. In transportation, graph analytics can enable personalized and stochastic routing that takes into account time-varying travel times and greenhouse gas emissions. These are but some uses of graph data analytics; new ones emerge at an accelerated pace.

Practical graph analytics uses a combination of graph-theoretic, statistical, and data management techniques to model, store, query, and analyze graph data. The processing of graph data embodies computationally hard tasks that have only been exacerbated by the rapid and increasing growth in available graph-related data. In 2010, Google introduced Pregel, a distributed system for large-scale graph processing. Inspired by the Bulk Synchronous Parallel model, Pregel features an intuitive, vertex-centric organization of computations that lets its users "think like a vertex". Since its introduction, Pregel has spurred substantial interest in large-scale graph data processing, and a number of Pregel-like systems have emerged.

Apache Giraph is an open-source system for Pregel-like, large-scale graph data processing. It has a global and growing user community and is thus an increasingly popular system for managing and analyzing graph data. This book provides a step-by-step guidance to data management professionals, students, and researchers who are looking to understand and use Apache Giraph. It guides the reader through the details of installing and configuring the system. It offers a detailed description of the programming model and related aspects. And it offers a step-by-step

coverage of the implementation of several popular, as well as advanced, graph analytics algorithms, covering also related optimization details.

In a nutshell, this book is a timely and valuable resource for data management professionals, students, and researchers interested in using Apache Giraph for large-scale graph processing and graph analytics.

Aalborg, Denmark Christian S. Jensen
July 2016

Preface

We are generating data more than ever. The ubiquity of the Internet has dramatically changed the size, speed, and nature of the generated data. Almost every human became a data generator and every business became a digital business. As a result, we are witnessing a data explosion. In the past few years, several technologies have contributed to this data explosion including mobile computing, Web 2.0, social media, social network, cloud computing and Software-as-a-Service (SaaS). In the future, it is expected that the Internet of Things will further amplify this challenge. In particular, several *things* would be able to get connected to the Internet, and thus there will be lots of data passed from users to devices, to servers, and back. Hence, in addition to the billions of people who are currently using the Internet and daily producing a lot of data, watches, cars, fridges, toaster, and many other devices will be online and continuously generating data as well. It is quite expected that in the near future, our toasters will be able to recommend types of bread based on suggested information from our friends on the social networks.

With the recent emerging wave of technologies and applications, the world has become more connected than ever. Graph is a popular neat data structure which is used to model the data as an arbitrary set of objects (vertices) connected by various kinds of relationships (edges). With the tremendous increase in the size of the graph-structured data, large-scale graph-processing systems have been crucially on demand and attracted a lot of interest. This book is intended to take you to a journey with **Apache Giraph**, a popular distributed graph-processing platform, which is designed to bring the power of big data processing to graph data that would be too large to fit on a single machine. We describe the fundamental abstractions of the system and its programming models and describe various techniques for using the system to process graph data at scale. The book is designed as a self-study step-by-step guide for any reader with an interest in large-scale graph processing. All the source codes presented in the book are available for download from the associated Github repository of the book.

Organization of the Book

Chapter 1 starts with a general background of the big data phenomena. We then introduce the big graph problem, its applications, and how it differs from the traditional challenges of the big data problem and motivates the need for domain-specific systems that are designed to tackle the large-scale graph-processing problem. We then introduce the Apache Giraph system, its abstraction, programming model, and design architecture to set the stage for the reader and provide him with the fundamental information which is required to smoothly follow the other chapters of the book.

Chapter 2 takes Giraph as a platform. Keeping in view that Giraph uses Hadoop as its underlying execution engine, we explain how to set up Hadoop in different modes, how to monitor it, and how to run Giraph on top of it using its binaries or source code. We then move to explaining how to use Giraph. We start by running an example job in different Hadoop modes and then approach more advanced topics such as monitoring Giraph application life cycle and monitoring Giraph jobs using different methods. Giraph is a very flexible platform and its behavior can be tuned in many ways. We explain the different methods of configuring Giraph and end the chapter by giving a detailed description of setting up a Giraph project in Eclipse and IntelliJ IDE.

Chapter 3 provides an introduction to Giraph programming. We introduce the basic Giraph graph model and explain how to write a Giraph program using the vertex similarity algorithm as a use case. We explain three different ways of writing the driver program and their pros and cons. For loading data into Giraph, it comes packaged with numerous input formats for reading different formats of data. We describe each of the formats with examples and end the chapter with the description of Giraph output formats.

Chapter 4 discusses the implementation of some popular graph algorithms including PageRank, connected components, shortest paths, and triangle closing. In each of these algorithms, we give an introductory description and show some of its possible applications. Then using a sample data graph, we show how the algorithm works. Finally, we describe the implementation details of the algorithm in Giraph.

Chapter 5 sheds light on the advanced Giraph programming. We start by discussing common Giraph algorithmic optimizations and how those optimizations may improve the performance and flexibility of the algorithms implemented in Chap. 4. We explain different graph optimizations to enable users to implement complex graph algorithms. Then, we discuss a set of tunable Giraph configurations that controls Giraph's utilization of the underlying resources. We also discuss how to change Giraph's default partitioning algorithm and how to write a custom graph input and output format. We then talk about common Giraph runtime errors and finalize the chapter with information on Giraph's failure recovery.

Recently, several systems have been introduced to tackle the challenge of large-scale graph processing. In Chap. 6, we highlight two of these systems, GraphX and GraphLab. We describe their program abstractions and their programming models. We also highlight the main commonalities and differences between these systems and Apache Giraph.

Target Audience

We hope this book serves as a useful reference for students, researchers, and practitioners in the domain of large-scale graph processing.

To Students: We hope that the book provides you an enjoyable introduction to the field of large-scale graph processing. We have attempted to properly describe the state of the art and present the technical challenges in depth. The book will provide you with a comprehensive introduction and hands-on experience to tackling large-scale graph-processing problem using the Apache Giraph systems.

To Researchers: The material of this book will provide you with a thorough coverage for the emerging and ongoing advancements on big graph-processing systems. You also can use this book as a starting point to tackle your next research challenge in the domain of large-scale graph processing.

To Practitioners: You will find this book a very useful step-by-step guide with several code examples, with source codes available in the Github repository of the book, and programming optimization techniques so that you can immediately put the gained knowledge from this book into practice due to the open-source availability of Apache Giraph system.

Sydney, Australia — Sherif Sakr
Aalborg, Denmark — Faisal Moeen Orakzai
Thuwal, Saudi Arabia — Ibrahim Abdelaziz
Thuwal, Saudi Arabia — Zuhair Khayyat

Acknowledgments

I would like to thank my parents for their encouragement and support. I want to thank my children, Jana and Shehab, for the happiness and enjoyable moments they are always bringing to my life. My most special appreciation goes to my wife, Radwa Elshawi, for her everlasting support and deep love.

Sherif Sakr

I want to thank my father Moeen, for allowing me to follow my ambitions throughout my life and my mother Hikmat for wishing the best for me at every step. I want to thank Sana, for her constant love and support in spite of all the time it kept me away and to my son, Ibrahim, a recent addition to the family who is always making me smile.

Faisal Moeen Orakzai

I am filled with gratitude to my wife for her encouragement, dedication, and support. I am also grateful to my daughters for their love and the great time I spend with them.

Ibrahim Abdelaziz

I would like to thank my parents, Yarub and Nadia, for their endless encouragement and support. I would like to also thank my once-in-a-life-time love, Manal, for standing beside me throughout my studies and writing this book. Special thanks to my two kids, Mazin and Ziyad, for their enthusiasm and keeping me awake at night.

Zuhair Khyyat

Contents

1 Introduction 1
- 1.1 Big Data Problem 1
- 1.2 A Star Is Born: The MapReduce/Hadoop Framework 4
- 1.3 From Big Data to Big Graphs 9
- 1.4 Use Cases for Big Graphs 11
  - 1.4.1 Social Networks 11
  - 1.4.2 Web Graph 11
  - 1.4.3 Knowledge Bases and Linked Data 14
  - 1.4.4 Road Networks and Location-Based Services 16
  - 1.4.5 Chemical and Biological Networks 17
- 1.5 Graph Databases 17
- 1.6 Does Hadoop Work Well for Big Graph Processing? 19
- 1.7 BSP Programming Model and Google Pregel 22
- 1.8 Pregel Extensions 24
- 1.9 Giraph: BSP + Hadoop for Graph Processing 27
- 1.10 Book Roadmap 31
- 1.11 How to Use the Code Examples of This Book? 32

2 Installing and Getting Giraph Ready to Use 35
- 2.1 Installing Hadoop 35
  - 2.1.1 Single-Node Local Mode Installation 36
  - 2.1.2 Single-Node Pseudo-Distributed Installation 38
  - 2.1.3 Multi-node Cluster Installation 43
- 2.2 Monitoring Hadoop 46
  - 2.2.1 Hadoop Web User Interfaces 46
- 2.3 Installing Giraph 50
- 2.4 Installing Giraph from Source Code 51
- 2.5 Running an Example Giraph Job 53
  - 2.5.1 Hadoop Local Mode 53
  - 2.5.2 Pseudo-Distributed and Clustered Hadoop Mode 55
- 2.6 Monitoring Giraph Application Life Cycle 56
  - 2.6.1 `MasterObserver` 57
  - 2.6.2 `WorkerObserver` 57

2.7 Monitoring Giraph Jobs . . . 57
2.7.1 Using Hadoop Commands to Monitor Giraph Jobs . . . 58
2.7.2 Using Hadoop UI to Monitor Giraph Jobs . . . 62
2.8 Configuring Giraph . . . 62
2.8.1 Giraph-Specific Options . . . 62
2.8.2 Job-Specific Options . . . 63
2.8.3 Algorithm-Specific Options . . . 65
2.8.4 Giraph Configuration Summary . . . 66
2.9 Creating a Giraph IDE Project . . . 66
2.9.1 Eclipse . . . 68
2.9.2 Eclipse with Maven . . . 73
2.9.3 IntelliJ IDE . . . 76
2.9.4 IntelliJ IDE with Maven . . . 79
2.10 Debugging Local Zookeeper . . . 81

3 Getting Started with Giraph Programming . . . 87
3.1 Giraph Graph Model . . . 87
3.1.1 The `Edge` Interface . . . 87
3.1.2 The `Vertex` Interface . . . 88
3.1.3 The `Computation` Interface . . . 91
3.2 Vertex Similarity Algorithm . . . 93
3.2.1 Implementation in Giraph . . . 95
3.3 Writing a Basic Giraph Job . . . 98
3.4 Writing the Driver Program . . . 99
3.4.1 Using `main` Method . . . 101
3.4.2 Using `Tool` Interface . . . 101
3.4.3 Using `GiraphRunner` Class . . . 103
3.5 Preparing Graph Data for Giraph Input . . . 105
3.5.1 Hadoop Distributed File System . . . 105
3.5.2 Hadoop Input Formats . . . 105
3.5.3 Giraph Input Formats . . . 106
3.5.4 Giraph Vertex Input Formats . . . 107
3.5.5 Edge Input Formats . . . 113
3.6 Preparing Graph Data for Giraph Output . . . 113
3.6.1 Vertex Output Formats . . . 113
3.6.2 Edge Output Formats . . . 116

4 Popular Graph Algorithms on Giraph . . . 119
4.1 PageRank . . . 119
4.1.1 Example . . . 120
4.1.2 Implementation Details . . . 122
4.2 Connected Components . . . 123
4.2.1 Example . . . 123
4.2.2 Implementation Details . . . 124

4.3 Shortest Path . . . 127
4.3.1 Example . . . 127
4.3.2 Implementation Details . . . 129
4.4 Triangle Closing . . . 131
4.4.1 Example . . . 131
4.4.2 Implementation Details . . . 133
4.5 Maximal Bipartite Graph Matching . . . 135

5 Advanced Giraph Programming . . . 141
5.1 Algorithm Optimization . . . 141
5.1.1 MasterCompute . . . 141
5.1.2 Data Sharing Across Nodes . . . 143
5.1.3 Combiners . . . 148
5.1.4 Coarse-Grained Processing . . . 149
5.2 Dealing with More Complex Graph Algorithms . . . 150
5.2.1 Graph Mutations . . . 150
5.2.2 Undirected Graphs . . . 151
5.2.3 Synchronizing the States of Neighboring Vertices . . . 154
5.2.4 Implementing Graph Coloring . . . 156
5.3 Performance Optimizations . . . 162
5.3.1 Multithreading . . . 162
5.3.2 Message Exchange Tuning . . . 163
5.3.3 Controlling the OutEdges Class . . . 163
5.3.4 Out-of-Core Processing . . . 164
5.4 Giraph Custom Partitioner . . . 166
5.5 Advanced Giraph I/O . . . 166
5.5.1 Writing a Custom Input Format . . . 167
5.5.2 Writing a Custom Output Format . . . 168
5.6 Analyzing Giraph Errors . . . 169
5.6.1 CountersExceededException . . . 170
5.6.2 ClassNotFoundException . . . 170
5.6.3 FileAlreadyExistsException . . . 170
5.6.4 OutOfMemoryError . . . 171
5.7 Failure Recovery . . . 171
5.7.1 Checkpointing . . . 171
5.7.2 Retrying and Recovering Failed Jobs . . . 172

6 Related Large-Scale Graph Processing Systems . . . 175
6.1 GraphX . . . 175
6.1.1 Spark and RDD . . . 175
6.1.2 GraphX RGD . . . 179
6.1.3 Examples . . . 183
6.2 GraphLab . . . 189
6.2.1 Programming Model . . . 189
6.2.2 Consistency Model . . . 194

References . . . 195

About the Authors

Sherif Sakr is currently Professor of Computer and Information Science in the Health Informatics Department at King Saud bin Abdulaziz University for Health Sciences. He is also affiliated with the University of New South Wales and DATA61/CSIRO (formerly NICTA). He received his Ph.D. degree in Computer and Information Science from Konstanz University, Germany in 2007. He received his B.Sc. and M.Sc. degrees in Computer Science from the Information Systems Department at the Faculty of Computers and Information in Cairo University, Egypt, in 2000 and 2003, respectively. In 2008 and 2009, Sherif held an Adjunct Lecturer position at the Department of Computing of Macquarie University. He also held visiting appointments in several academic and research institutes including Microsoft Research (2011), Alcatel-Lucent Bell Labs (2012), Humboldt University of Berlin (2015), University of Zurich (2016), and TU Dresden (2016). In 2013, Sherif has been awarded the Stanford Innovation and Entrepreneurship Certificate.

Faisal Moeen Orakzai is a joint Ph.D. candidate at Université Libre de Bruxelles (ULB) Belgium and Aalborg University (AAU) Denmark. He received his joint Masters degree in Computer Science from ULB and TU Berlin in 2014. After graduating as a computer engineer in 2006 from National University of Sciences and Technology (NUST), he worked in the enterprise software development industry for a couple of years before getting attracting towards distributed and large-scale data processing systems. In addition to research, he works as a consultant and helps companies setting up their distributed data processing architectures and pipelines. He is a big data management and analytics enthusiast and currently working on a Giraph based framework for spatio-temporal pattern mining.

Ibrahim Abdelaziz is a Computer Science Ph.D. candidate at King Abdullah University of Science and Technology (KAUST). He received his B.Sc. and M.Sc. degrees from Cairo University in 2007 and 2012, respectively. Prior to joining KAUST, he used to work on pattern recognition and information retrieval in several research organizations in Egypt. His current research interests include data mining over large-scale graphs, distributed systems, and machine learning.

Zuhair Khayyat is a Ph.D. candidate focusing on big data, analytics, and graphs in the InfoCloud group at King Abdullah University of Science and Technology (KAUST). He also received his Masters degree in Computer Science from KAUST in 2010. He received his B.Sc. degree in Computer Engineering in 2008 from King Fahd University of Petroleum and Minerals (KFUPM).

List of Figures

Figure 1.1 3V characteristics of big data. 3
Figure 1.2 An example MapReduce program [1]. 5
Figure 1.3 An overview of the flow of execution a MapReduce operation [1] 7
Figure 1.4 Execution steps of the word count example using the MapReduce framework. 8
Figure 1.5 Graph-based modeling 10
Figure 1.6 Snippet of an example of directed social graph 12
Figure 1.7 Snippet of an example undirected social graph 13
Figure 1.8 Structure of Web Graph 14
Figure 1.9 Linked open data collection 15
Figure 1.10 Location-based services 16
Figure 1.11 Biological networks 18
Figure 1.12 MapReduce iteration 20
Figure 1.13 BSP programming model 23
Figure 1.14 Vertex-based programming model 23
Figure 1.15 Supersteps execution in BSP. 24
Figure 1.16 Vertex voting in BSP. 24
Figure 1.17 BSP execution for iterative processing 25
Figure 1.18 Layered architecture of Giraph 28
Figure 1.19 Pregel and Giraph 28
Figure 1.20 Zookeeper synchronization 28
Figure 1.21 BSP example 29
Figure 1.22 Execution flow of Giraph 30
Figure 1.23 Giraph versus graph databases. 31
Figure 2.1 HDFS Web UI 47
Figure 2.2 HDFS live nodes page 48
Figure 2.3 DataNode Web UI. 48
Figure 2.4 MapReduce cluster summary. 49
Figure 2.5 MapReduce cluster summary-bad nodes 49
Figure 2.6 MR Web UI: active task trackers. 49
Figure 2.7 Giraph download options 50
Figure 2.8 Giraph application life cycle 56

Figure 2.9 Output of `hadoop job -list` command 60
Figure 2.10 Output of `hadoop job -status` command 60
Figure 2.11 Output of `hadoop job -history <job-output-dir>` command 61
Figure 2.12 Giraph job details 63
Figure 2.13 Running map tasks 64
Figure 2.14 Task attempts 64
Figure 2.15 Giraph configuration hierarchy 64
Figure 2.16 Eclipse: new Java project window 71
Figure 2.17 Eclipse: new Java project settings window 72
Figure 2.18 Eclipse: build path context menu 73
Figure 2.19 Eclipse: adding external Jars 74
Figure 2.20 Eclipse: creating new Java class. 75
Figure 2.21 Eclipse: new Java class window 77
Figure 2.22 Eclipse: conversion to Maven project 78
Figure 2.23 Eclipse: dependencies addition. 78
Figure 2.24 IntelliJ IDEA: new project window with JDK selection 79
Figure 2.25 IntelliJ IDEA: new project template selection window 80
Figure 2.26 IntelliJ IDEA: new project name window 81
Figure 2.27 IntelliJ IDEA: giraph project structure 82
Figure 2.28 IntelliJ IDEA: project structure window 83
Figure 2.29 IntelliJ IDEA: attaching Jar files 83
Figure 2.30 Add framework support to Intellij Java project 85
Figure 2.31 Intellij: Add Maven support to the project. 85
Figure 2.32 Intellij: enableauto-import of projectdependencies 86
Figure 3.1 `Edge<I extends WritableComparable, E extends Writable>` 88
Figure 3.2 `Vertex<I extends WritableComparable, V extends Writable, E extends Writable>` 88
Figure 3.3 Social graph 93
Figure 3.4 Vertex similarity: superstep 0 94
Figure 3.5 Vertex similarity: superstep 1 94
Figure 3.6 Basic job composition 98
Figure 3.7 Ways of writing a driver program 99
Figure 3.8 Tiny graph (giraph.apache.org) 106
Figure 3.9 Graph conversion by `LongDoubleDoubleTextInputFormat`. 109
Figure 3.10 Graph conversion by `SccLongLongNullTextInputFormat`. 109
Figure 3.11 Graph conversion by `NormalizingLongDoubleDoubleTextInputFormat` 112
Figure 4.1 Sample web graph: Nodes are web pages while edges represent the outgoing links between them 120

Figure 4.2 Running PageRank algorithm for four supersteps using the data graph shown in Fig. 4.1 121
Figure 4.3 Sample social graph: Users associated with their interests 123
Figure 4.4 Evaluating connected components algorithm using the social graph shown in Fig. 4.3. 125
Figure 4.5 Sample road network: Nodes represent cities while edges represent the roads between them. Edge labels denote the cost to travel from one city to another 127
Figure 4.6 Evaluating Single Source Shortest Path (SSSP) algorithm using the sample road network in Fig. 4.5 starting from Chicago city 128
Figure 4.7 Sample social graph: Nodes represent users while edges represent their friendship connections 132
Figure 4.8 Evaluating Triangle Closing algorithm using the sample social network in Fig. 4.7 132
Figure 5.1 MasterCompute 142
Figure 5.2 Simple aggregator 147
Figure 5.3 A directed graph with two outgoing edges from vertex 2 . . . 152
Figure 5.4 An undirected graph with two edges 152
Figure 5.5 A directed graph that best describes within Giraph the graph in Fig. 5.4 152
Figure 5.6 Example figure 154
Figure 5.7 Superstep 1: Value assignment 155
Figure 5.8 Superstep 2: Collision detection between vertices B and E 155
Figure 5.9 Supersteps 3–5: Collision resolve; vertex E in back-off period. 155
Figure 5.10 Supersteps 6: New assignment for vertex E. 156
Figure 5.11 Example directed graph of courses with common students 157
Figure 5.12 Example undirected graph of courses with common students 157
Figure 5.13 Superstep 1: Color assignment. 158
Figure 5.14 Superstep 2: Collision detection; all vertices revert to random back-off state 158
Figure 5.15 Superstep 3: Vertex CS360 becomes active 159
Figure 5.16 Superstep 4: Vertices CS201 and CS261 become active 159
Figure 5.17 Superstep 5: Vertices CS202 and CS252 become active 160
Figure 5.18 Superstep 6: All vertices are done with color assignment . . . 160
Figure 6.1 Spark framework versus hadoop framework 176
Figure 6.2 Flow of RDD operations in spark 178
Figure 6.3 Spark's ecosystem 180
Figure 6.4 RDG representation of graphs in graphX 181

Figure 6.5 Unified representation of graphs in graphX 182
Figure 6.6 Word-text-subject graph 184
Figure 6.7 An example graph with vertex IDs used as vertex values . . . 190
Figure 6.8 The result of undirected connected components on the example graph in Fig. 6.7 with vertex IDs used as vertex values 190
Figure 6.9 Edge-Cut versus vertex-cut partitioning schemes 192
Figure 6.10 GraphLab consistency models 193

List of Tables

Table 2.1 Giraph dependency options 53
Table 2.2 Sample Giraph job parameters 54
Table 2.3 A summary of Giraph's configurations described in this book 67
Table 5.1 A comparison between different out-degree edges storages. . . . 164
Table 6.1 Sample spark's transformations 178
Table 6.2 Sample spark's actions 178