UNIVERSITY OF NICE - SOPHIA ANTIPOLIS

These d'Habilitation a Diriger des Recherches

presented by

Pietro MICHIARDI

Scalable Data Management Systems: distribution, storage and processing of large amounts of data

Submitted in total fulfillment of the requirements of the degree of Habilitation a Diriger des Recherches

Committee :

Acknowledgments

This manuscript contains a collection of research works that I had the chance to carry out with many bright students. As a junior faculty at first, and with a more “mature” approach at last, I enjoyed spending countless hours discussing about what constitute good research questions (that is how to solve the right problem, not how to solve a problem right), coding algorithms, simulators and more elaborate pieces of software, working through complex mathematical models and (finally!) applying all the lessons learned on statistics on endless measurement traces.

All in all, writing papers – well, good papers – remained the most complex exercise myself, and the students working with me, ever had to deal with. Enjoying notifications of acceptance, and sharing the difficult times related to a rejection, permeated the entire lapse of time covered in this manuscript. However, I distinctly remember that both good and bad news were always accepted and dealt with a positive attitude, aiming at a constant improvement of our technical skills and our ability to “package” our work for the next round of reviews.

For all this, I wish to thank all the Ph.D. students that I supervised, and all my colleagues – both in my team, and in other research centers or universities.

I’d like to thank my family for the moral support, and for pushing me to complete this manuscript: my parents, for being always present and caring, my wife, for finding the courage to listen to my rehearsals, my (research) problems and for being so supportive, and my kids, for animating my sleepless nights!

I hope that, with a brand new accreditation for my ability to conduct and supervise research works, the journey I started a long time ago will continue at the same pace, with the same curiosity and with the same ambition for producing quality and useful work!

Contents

CHAPTER 1

Curriculum Vitae

1.1 Appointments

2009 – **Assistant Professor, class 1**, EURECOM, Sophia-Antipolis, France
2005 – 2009 **Assistant Professor, class 2**, EURECOM, Sophia-Antipolis, France
2001-2004 **Ph.D. Student**, Telecom ParisTech / EURECOM
2000-2001 **Research Engineer**, EURECOM, Sophia-Antipolis, France

1.2 Education

2004: **Ph.D.**, Computer Science (honors), Telecom Paris Tech, France
2001: **M.Sc.**, Electrical Engineering (honors), Politecnico di Torino, Italy
1999: **M.Sc.**, Computer Science, EURECOM, France

1.3 Teaching and Academic Activities

- **Large-scale Distributed Systems and Cloud Computing**, 2012–, Professor
- **Applied Algorithm Design**, 2007–, Professor
- **Applied Game Theory**, 2009–2013, Professor
- **Web Technologies**, 2005–2011, Professor

Coordinator of the Double-Degree M.Sc. Programme Politecnico di Torino / Telecom ParisTech
Coordinator of the Joint Doctoral School Politecnico di Torino / Eurecom

1.4 Supervision

PhD Students

- **Hung-Duy Phan**, Ph.D student (Telecom Paris Tech, France), 2013–, Thesis title: “High-level Languages for Data-intensive Scalable Computing and their Optimization”

- **Mario Pastorelli**, Ph.D student (Telecom Paris Tech, France), 2011–, Thesis title: "Theory and Practice of Job Scheduling on Parallel Machines"
- **Xiaolan Sha**, Ph.D student (Telecom Paris Tech, France – CIFRE scholarship), 2010–2013, Thesis title: "Scalable Recommender Systems for Trend Prediction"
- **Laszlo Toka**, Ph.D student (Telecom Paris Tech, France – Budapest University of Technology and Economics, Hungary), 2007-2011, Thesis title: "Large-scale Distributed Backup and Storage Systems", *now Research Fellow at Ericsson Labs, Hungary*
- **Chi-Anh La**, Ph.D. student (Telecom Paris Tech, France), 2006-2010, Thesis title: "Content Distribution Systems for Mobile Networks", *now PostDoc Fellow at Grenoble Informatics Laboratory, France*

Post Doctoral Researchers and Research Engineers

- **Xiaolan Sha**, Post-Doctoral Fellow (PhD Telecom ParisTech)
- **Daniele Venzano**, Research engineer (M.S. University of Genova)
- **Antonio Barbuzzi**, Post-Doctoral Fellow (Politecnico di Bari, Italy), now Full-time Researcher
- **Matteo Dell'Amico**, Post-Doctoral Fellow (PhD Univ. of Genova, Italy), now Full-time Researcher
- **Damiano Carra**, Post-Doctoral Fellow (PhD Univ. of Trento, Italy), "P2P Content Distribution Systems", now Ass. Prof. at University of Verona

Visiting PhD Students

- **Antonio Barbuzzi**, Visiting Ph.D. student (Politecnico di Bari, Italy), 2010, "Large-scale Distributed Systems for Analytics Applications"
- **Flavio Esposito**, Visiting Ph.D. student (Boston University, USA), 2007, "P2P Content Distribution Applications"
- **Krishna Ramachandran**, Visiting Ph.D. student (Rensselaer Polytechnic Institute – RPI, USA), 2006, "P2P Content Distribution Applications"

1.5 Participation as a Ph.D. Jury Member

Giulia Mega, Ph.D. University of Trento
Michelle Wettervald, Ph.D. Telecom ParisTech
Anna-Kaisa Pietilainen, Ph.D. Universite Paris VI Pierre et Marie Curie
Roberto Cascella, Ph.D. University of Trento
Jose Costa-Requena, Ph.D. Technical University of Helsinky

1.6 Funded Research Projects

EU Funded Projects:

- IST-STREP FP7, *BIGFOOT*, (2012-2015): *Project Coordinator*, Big Data Analytics of Digital Footprints, http://bigfootproject.eu
- IST-IP FP7, *MPLANE*, (2012-2015): Intelligent Measurement Plane for Future Network and Application Management, http://www.ict-mplane.eu
- IST-STREP FP7, *NADA*, (2009-2011): Edge capacity hosting overlays of NAno DAta centers, http://www.nanodatacenters.eu
- IST-FET FP6, *CASCADAS* (2006-2008): Component-ware for Autonomic Situation-aware Communications, and Dynamically Adaptable Services, http://www.cascadas-project.org
- IST-FET FP5, *MOBILEMAN* (2002-2004): Mobile Metropolitan Ad hoc Networks, http://cnd.iit.cnr.it/mobileMAN

French-Government Funded Projects:

- ANR, *PROSE*, (20010-2013): Content Shared Through Peer-to-Peer Recommendation & Opportunistic Social Environment
- ANR, *VIPEER*, (20010-2013): Video Traffic Engineering in an Intra-Domain Context using Peer-to-Peer Paradigms
- ANR-A*STAR, MERLION project (SINGAPORE) (2009-2010): Peer-to-peer Content Storage
- ANR, *SPREADS*, (2008-2010): Safe P2p RealiablE Architecture for Data Storage, http://www.spreads.fr
- GET, *DISPAIRSE* (2007-2008): Disque Pair a Pair Securise, https://dispairse.point6.net/index.php/Accueil
- ACI, *SPLASH* (2003-2005): Securing mobile ad hoc networks, http://www.inrialpes.fr/planete/splash.html

1.7 Honors and Awards

- *Best paper* award at the 1st International Workshop on Optimization in Wireless Networks (WiOpt 2003) Sophia Antipolis, France for the paper entitled “Game Theoretic Analysis of Cooperation Enforcement in MANET”, co-authored with R. Molva.
- *Telecom Valley Innovation Prize* (prix de l’innovation), 2001, Sophia Antipolis, France for the Patent “Process for providing non repudiation of receipt (NRR) in an electronic transaction environment”.

- *Award scholarship* for the academic year 1998-1999 from the European Community, Erasmus Programme.

1.8 Publications

Book Chapters:

1. *Michiardi, Pietro*; Molva, Refik, "Ad hoc Networks Security", *Handbook of Information Security, ISBN: 0-471-64833-7, John Wiley & Sons, Hardcover: 3 Volumes, December 2005 , Vol. 2*

2. *Michiardi, Pietro*; Molva, Refik, "Ad hoc Networks Security", *Mobile Ad Hoc Networking, ISBN: 0471373133, John Wiley & Sons, Hardcover: 416 pages, July 2004 , pp 329-354*

Referred Journals:

3. Carra, Damiano; Steiner, Moritz; *Michiardi, Pietro*; Biersack, Ernst W; Effelsberg, Wolfgang; En-Najjary, Taoufik, "Characterization and management of popular content in KAD", To appear in *IEEE* **Transactions on Parallel and Distributed Systems**

4. Carra, Damiano; *Michiardi, Pietro*; Salah, Hani; Strufe, Thorsten, "On the impact of incentives in eMule, analysis and measurements of a popular file-sharing application", To appear in *IEEE Journal on Selected Areas in Communications (***JSAC***)- Special Issue on Emerging Technologies in Communications*

5. *Michiardi, Pietro*; Carra, Damiano; Albanese, Francesco; Bestavros, Azer, "Peer-assisted Content Distribution on a Budget", *Computer Networks, Elsevier, February 2012*

6. La, Chi Anh; *Michiardi, Pietro*; Casetti, Claudio E; Chiasserini, Carla-Fabiana; Fiore, Marco, "Content Replication in Mobile Networks", *IEEE Journal on Selected Areas in Communications (***JSAC***), Special Issue Cooperative Networking Challenges and Applications, Volume 30, N.9, October 2012*

7. Esposito, Flavio; Matta, Ibrahim; Bera, Debajyoti; *Michiardi, Pietro*, "On the impact of seed scheduling in peer-to-peer networks", *Computer Network, Elsevier, October, 2011*

8. Carra, Damiano; Neglia, Giovanni; *Michiardi, Pietro*; Albanese, Francesco, "On the Impact of Greedy Strategies in BitTorrent Networks: The Case of BitTyrant", *IEEE* **Transactions on Parallel and Distributed Systems**, *2011*

9. Kato, Daishi; Elkhiyaoui, Kaoutar; Kunieda, Kazuo; Yamada, Keiji; *Michiardi, Pietro*, "A scalable interest-oriented peer-to-peer Pub/Sub network", *Peer-to-Peer Networking and Applications Journal, Springer, 2011*

10. Toka, Laszlo; *Michiardi, Pietro*, "Analysis of User-driven Peer Selection in Peer-to-Peer Backup and Storage Systems", *Telecommunication Systems Journal, 2011*

11. Smaragdakis, Georgios; Laoutaris, Nikolaos; *Michiardi, Pietro*; Bestavros, Azer; Byers, John W.; Roussopoulos, Mema. "Distributed Network Formation for n-way Broadcast Applications", *IEEE* **Transactions on Parallel and Distributed Systems***, 2010*

12. Di Pietro, Roberto; *Michiardi, Pietro*; Molva, Refik, "Confidentiality and integrity for data aggregation in WSN using peer monitoring", *Security and Communication Networks Journal, Wiley, 2009*

13. Bagga, Walid; Crosta, Stefano; *Michiardi, Pietro*; Molva, Refik, "Establishment of ad-hoc communities through policy-based cryptography", *in Elsevier Electronic Notes in Theoretical Computer Science, Volume 171, N. 1, pp 107-120, 2007*

14. *Michiardi, Pietro*; Molva, Refik, "Analysis of Coalition Formation and Cooperation Strategies in Mobile Ad hoc Networks", *Ad Hoc Networks, Volume 3, N 2, March 2005 , pp 193-219*

15. *Michiardi, Pietro*; Molva, Refik, Contribution to "Working Session on Security in Ad Hoc Networks", *ACM SIGMOBILE Mobile Computing and Communications Review, Volume 7, N. 1, 2003*

16. *Michiardi, Pietro*; Molva, Refik, "Ad hoc Network Security", *ST Journal of System Research, Volume 4, N. 1, March 2003*

Referred International Conferences and Workshops:

17. Sha, Xiaolan; Quercia, Daniele; Dell'Amico, Matteo; *Michiardi, Pietro*, "Trend Makers and Trend Spotters in a Mobile Application", *in Proc. of* **ACM CSCW***, 2013*

18. Lauinger, Tobias; Kirda, Engin; *Michiardi, Pietro*, "Paying for piracy? An analysis of one-click hosters' controversial reward schemes", *in Proc. RAID, 2012*

19. Sha, Xiaolan; Quercia, Daniele; Dell'Amico, Matteo; *Michiardi, Pietro*, "Spotting trends: The wisdom of the few", *in Proc. of* **ACM RECSYS***, 2012*

20. Mager, Thomas; Biersack, Ernst; *Michiardi, Pietro*, "A measurement study of the Wuala on-line storage service", *in Proc. of* **IEEE P2P***, 2012*

21. Toka, Laszlo; Cataldi, Pasquale; Dell Amico, Matteo; *Michiardi, Pietro*, “Redundancy management for P2P backup”, *in Proc. of* **IEEE INFOCOM**, *2012*

22. Baer, Arian; Barbuzzi, Antonio; *Michiardi, Pietro*; Ricciato, Fabio, “Two parallel approaches to network data analysis”, *in Proc. of ACM LADIS, 2011*

23. Toka, Laszlo; Dell Amico, Matteo; *Michiardi, Pietro*, “Data transfer scheduling for P2P storage”, *in Proc. of* **IEEE P2P**, *2011*

24. Carra, Damiano; Steiner, Moritz; *Michiardi, Pietro*, “Adaptive load balancing in KAD”, *in Proc. of* **IEEE P2P**, *2011*

25. Sharma, Rajesh; Datta, Anwitaman; Dell Amico, Matteo; *Michiardi, Pietro*, “An empirical study of availability in friend-to-friend storage systems”, *in Proc. of* **IEEE P2P**, *2011*

26. Toka, Laszlo; Dell Amico, Matteo; *Michiardi, Pietro*, “Online data backup : a peer-assisted approach”, *in Proc. of* **IEEE P2P**, *2010*

27. Barbuzzi, Antonio; *Michiardi, Pietro*; Biersack, Ernst W; Boggia, Gennaro, “Parallel bulk Insertion for large-scale analytics applications”, *in Proc. of ACM LADIS, 2010*

28. Dell amico, Matteo; *Michiardi, Pietro*; Roudier, Yves, “Measuring password strength: an empirical analysis”, *in Proc. of* ***IEEE INFOCOM***, *2010*

29. La, Chi-Anh; *Michiardi, Pietro*; Casetti, Claudio; Chiasserini, Carla-Fabiana; Fiore, Marco, “A Lightweight Distributed Solution to Content Replication in Mobile Networks”, *in In Proc. of IEEE WCNC, 2010*

30. *Michiardi, Pietro*; Toka, Laszlo, “Selfish neighbor selection in peer-to-peer backup and storage applications ”, *in Proc. of* ***Euro-Par***, *2009*

31. Michiardi, Pietro; Chiasserini, Carla-Fabiana; Casetti, Claudio; La, Chi-Anh; Fiore, Marco, “On a selfish caching game”, *in Proc. of* ***ACM PODC***, *2009*

32. Elkhiyaoui, Kaoutar; Kato, Daishi; Kunieda, Kazuo; Yamada, Keiji; Michiardi , Pietro, “A Scalable Interest-oriented Peer-to-Peer Pub/Sub Network”, *in Proc. of* ***IEEE P2P***, *2009*

33. Esposito, Flavio; Matta,Ibrahim; *Michiardi, Pietro*; Nobuyuki, Mitsutake; Carra, Damiano, “Seed scheduling for peer-to-peer networks”, *in Proc. of IEEE NCA, 2009*

34. Toka, Laszlo; *Michiardi, Pietro*, “Uncoordinated Peer Selection in P2P Backup and Storage Applications”, *in Proc. of* ***IEEE INFOCOM***, *Global Internet Symposium, 2009*

35. Casetti, Claudio; Chiasserini, Carla-Fabiana; Fiore, Marco; La, Chi-Anh; *Michiardi, Pietro* ,"P2P Cache-and-Forward Mechanisms for Mobile Ad Hoc Networks", *in Proc. of IEEE ISCC 2009*

36. Laoutaris, Nikolaos; Carra, Damiano; *Michiardi, Pietro*, "Uplink Allocation Beyond Choke/Unchoke: or How to Divide and Conquer Best", *in Proc. of **ACM CoNEXT**, 2008*

37. Carra, Damiano; Neglia, Giovanni; *Michiardi, Pietro*, "On the Impact of Greedy Strategies in BitTorrent Networks: The Case of BitTyrant", *in Proc. of **IEEE P2P**, 2008*

38. Laszlo, Toka; *Michiardi, Pietro*, "Brief announcement : A dynamic exchange game", *in Proc. of **ACM PODC**, 2008*

39. Di Pietro, Roberto; *Michiardi, Pietro*, "Brief announcement : Gossip-based aggregate computation, computing faster with non address-oblivious schemes", *in Proc. of **ACM PODC**, 2008*

40. La, Chi-Anh ; *Michiardi, Pietro*, "Characterizing user mobility in Second Life", *in Proc. of **ACM SIGCOMM** Workshop WOSN, 2008*

41. Smaragdakis, Georgios ; Laoutaris, Nikolaos ; *Michiardi, Pietro* ; Bestavros, Azer ; Byers, John W; Roussopoulos, Mema "Swarming on optimized graphs for n-way broadcast", *in Proc. of **IEEE INFOCOM**, 2008*

42. *Michiardi, Pietro* ; Marrow, Paul ; Tateson, Richard ; Saffre, Fabrice, "Aggregation dynamics in service overlay networks", *in Proc. of IEEE/ACM SASO, 2007*

43. Oualha, Nouha ; *Michiardi, Pietro* ; Roudier, Yves, "A game theoretic model of a protocol for data possession verification", *in Proc. of IEEE WOWMOM Workshop TSPUC, 2007*

44. *Michiardi, Pietro* ; Ramachandran, Krishna ; Sikdar, Biplap, "Modeling seed scheduling strategies in BitTorrent", *in Proc. of **IFIP NETWORKING**, 2007*

45. *Michiardi, Pietro* ; Urvoy-Keller, Guillaume, "Performance analysis of cooperative content distribution for wireless ad hoc networks", *in Proc. of IEEE/IFIP WONS, 2007*

46. Legout, Arnaud ; Urvoy-keller, Guillaume ; *Michiardi, Pietro*, "Rarest first and choke algorithms are enough", *in Proc. of **ACM SIGCOMM/USENIX IMC**, 2006*

47. Bagga, Walid ; Crosta, Stefano ; *Michiardi, Pietro* ; Molva, Refik, "Establishment of ad-hoc communities through policy-based cryptography", *in Proc. of WCAN, Workshop on Cryptography for Ad hoc Networks, 2006*

48. *Michiardi, Pietro*, “Cooperation dans les reseaux ad-hoc: Application de la theorie des jeux et de l’evolution dans le cadre d’observabilité imparfaite”, *in Proc. of SSTIC 2006*

49. *Michiardi, Pietro* ; Molva, Refik, “Identity-based message authentication for dynamic networks”, *in Proc. of IFIP SEC, 2006*

50. Urvoy-Keller, Guillaume ; *Michiardi, Pietro*, “Impact of inner parameters and overlay structure on the performance of BitTorrent”, *in Proc. of **IEEE INFOCOM**, Global Internet Symposium, 2006*

51. Lavecchia, Claudio ; *Michiardi, Pietro* ; Molva, Refik, “Real life experience of Cooperation Enforcement Based on Reputation (CORE) for MANETs”, *in Proc. of IEEE ICPS Workshop Realman, 2005*

52. Altman, Eitan ; Kherani, Arzad A. ; *Michiardi, Pietro* ; Molva, Refik, “Non cooperative forwarding in ad hoc networks”, *in Proc. of **IFIP NETWORKING**, 2005*

53. Altman, Eitan ; Borkar, Vivek S. ; Kherani, Arzad A. ; *Michiardi, Pietro* ; Molva, Refik, “Some game-theoretic problems in wireless ad hoc networks”, *in Proc. of EURO-NGI, 2005*

54. Altman, Eitan ; Kherani, Arzad A. ; *Michiardi, Pietro* ; Molva, Refik, “Non cooperative forwarding in ad hoc networks”, *Accepted in IEEE PIMRC, 2004*

55. Molva, Refik ; *Michiardi, Pietro*, “Security in Ad hoc Networks”, *in Proc. of IFIP PWC, 2003*

56. *Michiardi, Pietro* ; Molva, Refik, “A game theoretic approach to evaluate cooperation enforcement mechanisms in mobile ad hoc networks”, *in Proc. of **IEEE WiOpt**, 2003*

57. *Michiardi, Pietro* ; Molva, Refik, “CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks”, *in Proc. of IFIP CMS, 2002*

58. *Michiardi, Pietro* ; Molva, Refik, “Simulation-based analysis of security exposures in mobile ad hoc networks”, *in Proc. of IFIP EW, 2002*

59. *Michiardi, Pietro* ; Molva, Refik, “Inter-domain authorization and delegation for business-to-business e-commerce”, *in Proc. of eBWC, 2001*

Tutorials, Ph.D. Schools and Invited Talks:

1. *Seminar* “The BigFoot project”, project presentation and research directions, ORANGE LABS, 2013

2. *Seminar* “The BigFoot project”, project presentation and research directions, AMADEUS, 2013
3. *Tutorial*, “Hot topics in BigData and Cloud Computing”, University of Verona, 2013
4. *Seminar*, “The BigFoot project”, project presentation and research directions, EDF R&D, 2013
5. *Tutorial*, “Data-intensive Computing with MapReduce”, Telecom ParisTech, FRANCE, 2012
6. *Tutorial*, “Data-intensive Computing with MapReduce”, TUB, GERMANY, 2011
7. *Tutorial*, “Data-intensive Computing with MapReduce”, NTNU, SINGAPORE, 2011
8. *Summer-School*, “Practical Problem Solving with Hadoop”, TUM, GERMANY, 2011
9. *Tutorial*, “Distributed Replication in Mobile Wireless Networks”, NTNU, SINGAPORE, 2010
10. *Seminar*, “Data-intensive Computing with MapReduce”, INRIA, FRANCE, 2010
11. *Seminar*, “Peer-to-peer Content Distribution”, Politecnico di Torino, ITALY, 2009
12. *Seminar*, “Peer-to-peer Content Distribution”, TUB, GERMANY, 2009
13. *Tutorial*, “Peer-to-peer Networks and Applications”, University of Piraeus, GREECE, 2008
14. *Tutorial*, “Greedy strategies in Peer-to-peer networks”, University of Athens, GREECE, 2008
15. *Invited Talk*, “P2P Content Distribution, the case of BitTorrent”, INRIA, FRANCE, 2008
16. *Invited Talk*, “Peer Assisted Content Distribution”, Gruppo nazionale Telecomunicazioni e Teoria dell’Informazione (GTTI), ITALY, 2008
17. *Invited Talk*, “Mining Second Life: Characterizing user mobility in Networked Virtual Environments”, Politecnico di Torino, ITALY, 2008
18. *Invited Talk*, “Mining Second Life: Characterizing user mobility in Networked Virtual Environments”, Universita di Catania, ITALY, 2008

19. *Invited Talk*, “Mining Second Life: Characterizing user mobility in Networked Virtual Environments”, Boston University Computer Science Colloquium Series, USA, 2008

20. *Invited Talk*, “Selfish Neighbor Selection in Service Overlay Networks”, Learning and Intelligent OptimizatioN, ITALY, 2007

21. *Invited Talk*, “Gossip-based aggregate computation: an optimal address non-oblivious scheme”, INRIA-EURECOM-THOMSON Seminar, FRANCE, 2007

22. *Invited Talk*, “Computational economics for self-organizing networks”, IST DISTRUST, BELGIUM, 2006

23. *Invited Talk*, “Non-cooperative load balancing for P2P service networks”, ESAS, GERMANY, 2006

24. *Invited Talk*, “Mecanismes de securite et de cooperation entre noeuds d’un reseaux mobile ad hoc”, Symposium sur la Securite des Technologies de l’Information et des Communications, FRANCE, 2006

25. *Tutorial*, “Ad hoc Networks Security”, Conference on Security and Network Architectures, FRANCE, 2004

26. *Invited Talk*, “Cooperation in Ad hoc Networks”, European Science Foundation, ITALY, 2002

Patent: Molva, Refik ; *Michiardi, Pietro*, “Process for providing non repudiation of receipt (NRR) in an electronic transaction environment”, Patent N°EP1300980

1.9 Professional Activities

TPC Member of several Conferences, among which: ACM SIGCOMM, ACM IMC, ACM MobiHoc, IEEE MASCOTS, IEEE P2P
Reviewer for several Conferences and Journals including: ACM SIGCOMM, IEEE INFOCOM, ACM CoNEXT, ACM CCS, ACM MobiHoc, ACM TISSEC, IEEE/ACM TON, IEEE ICNP, IEEE JSAC, IEEE TOMC, IEEE TPDS, IEEE TDSC, IEEE ICASSP
Local Chair: IEEE SecureComm 2007
TPC Co-Chair: IEEE WOWMOM Workshop TSPUC, 2006

CHAPTER 2

Introduction

The endeavor of this manuscript is to overview my work in the broad domain of *scalable data management systems*. As the title implies, the common thread to the research lines I will present in the following Chapters is the focus on the design of algorithms, their analysis and performance evaluation with the ultimate goal of building real systems or components thereof. In a general sense, the following Chapters address problems related to the management of data, including distribution (that is moving data over a computer network), storage (with several incarnations of storage systems which depend on data access patterns) and processing (*i.e.,* making sense of the data, which obviously depends on the application scenario at hand). A common characteristic that makes the above mentioned problems interesting is – in a wide sense – the system scale. For the works that target data distribution and storage, system scale (and dynamics) poses several challenges that call for the design of distributed algorithms capable of handling millions of hosts and manage their resources. For the works on systems used to process data, scalability is to be studied along two dimensions: the number of components of such systems (e.g. computing nodes) may be very large, and the volumes of data to be processed typically outweigh the capacity (storage and memory) of a single machine.

In addition, the techniques and methodology used to achieve the goals outlined above, also incorporate mathematical modeling and measurement campaigns, which are mainly required to overcome some limitations of a purely systems approach to research: in some cases, the scale of the systems under scrutiny implies that it simply not feasible to work on real working prototypes, thus one has to revert to models; yet, in other cases, it is important to understand the behavior of quantities (typically the input to those systems, including available resources and the behavior of eventual humans in the loop) that are not under the control of the system designer, which call for a thorough methodology to collect traces and replay them when assessing the performance of the system.

This manuscript is organized as follows. First, in this introductory Chapter, I overview three main topics that my group and I covered in the past years. My goal is to illustrate why I selected such topics, what are the important problems we addressed and to overview our approach to solve such problems. Then I will dedicate three Chapters to a more detailed, technical overview of the research lines covered in this manuscript: for the sake of brevity as well as clarity, the outline of all those chapter has a common theme. First, I start with an introduction to delineate the context of each work, then I spell out the problem statement, and finally I overview the solutions to the problem, which include a glance on the achieved results. The conclusion to each Chapter is a brief summary with highlights on the contributions, and open problems that have motivated other researchers to pursue the initial directions we established with our work.

Before proceeding with an overview of the contents of this manuscript, it is important to notice that I deliberately omit several works that I have produced on my own or with collaborators external to my group. Hence, in what follows I will focus solely on the contributions achieved by students and researchers in my group.

Chapter 3 is dedicated to the work I did with the first Ph.D. student (now graduated) under my supervision, Dr. Chi-Anh La. The topic of this line of research is **content replication** for mobile, wireless networks, in which we blend theory and practice relative to the design of distributed and scalable algorithms for the dissemination of data (*i.e.,* content residing in Internet servers) to a potentially large population of wireless devices, such as last-generation smart phones, in possession of individuals that move according to human-mobility patterns. This research work began just a few years after Internet Content Distribution peaked in popularity with the advent of applications such as BitTorrent. Peer-to-peer (P2P) applications for file sharing were already a fertile ground for research, and several important works in the domain [20,22,67,88,89,97] had appeared for the first time in the networking community. In addition, concurrently with the initial works of Chi-Anh, I developed invaluable experience in modeling, measuring real-life, deployed P2P applications, which led me to the definition of a new set of problems (in the area of distributed algorithms) which have been developed in the Ph.D. Thesis of Chi-Anh.

First, the network model we define in our work is substantially different from that used in the literature. A large fraction of works in the domain of Internet content dissemination assume whether a fixed infrastructure (the wired Internet) supporting both content and client or a completely decentralized wireless network in which content would be injected and disseminated by devices involved in operating the network. Our approach considers an hybrid network model in which content resides in Internet servers (on a fixed network) while the consumers of the content are equipped with both a cellular network interface and a wireless network interface in the family of the 802.11 standard. The first consequence of our network model is that content dissemination can be cast as a replication problem: users' devices replicate content that initially reside on Internet servers. Although content replication is not a new topic per-se, the constraints imposed by the new network model (which are well discussed in the Chapter dedicated to this line of work) makes it particularly challenging.

We thus study the problem of content replication and do so through the lenses of Location Theory of which, as a reference, the un-capacitated facility location problem is a simple and well-known incarnation. Our contribution is the design of a scalable, distributed algorithm for content replication, which we implemented and analyzed using a network simulation tool. In addition to the original context, in which we make the implicit assumption that all mobile terminals executing our algorithm would abide to its specification, we also studied the problem of content replication using non-cooperative game theory. Our main contribution is the definition of a new class of games that blend congestion and coordination games: we addressed the problem of finding equilibrium in such game (both in pure and mixed strategies) and arrived at the conclusion that (for a simple 2-player setting), the equilibrium strategy is closely related to the distributed algorithm we have defined for the cooperative version of the replication problem. This work has opened several new challenges that other groups extended in the past years. Finally, an original contribution of the work we present in Chapter 3 is that of defining an innovative measurement methodology to capture user mobility traces. Our work stems from the realization that performing measurement campaigns to study the behavior of humans while they move is a very hard problem, which has been tackled mainly by large research groups holding the necessary means to organize and deploy measurement probes. In contrast, we designed software tools to monitor movements of avatars in a popular (at the time) virtual world: our methodology makes the collection of mobility traces a trivial exercise, and our measurements illustrate (and validate) that mobility patterns in virtual worlds are, to a large extent, a very good match of what happens in the real world.

Chapter 4 is dedicated to the work I did with the second Ph.D. student (now graduated) under my supervision, Dr. Laszlo Toka. As it will become clear later on in the manuscript, this line of work generated a large number of research papers and software: for this I acknowledge the great work of Dr. Matteo Dell'Amico, who works in my group as a Post-Doctoral researcher. The focus of our work is scalable **content storage**, conceived both as an Internet application and an on-line service, offered by the communion of of end-host resources and possibly those of servers residing in large data centers. In this context, there are several problems that hinder the task of building a service that can guarantee data *availability* (that is, low-latency access to data) and *durability* (that is, negligible probability of data loss): storage and bandwidth resources (that at a large scale may appear to be unlimited) are not reliable because they are essentially bound to the on-line behavior of users.

When we first started our work in this domain, most of the research effort in the community was devoted to the study of new mathematical tools (specifically relevant to coding theory) to achieve data availability and durability in spite of failures, “churn"[1] and lack of a local copy of the original data, while minimizing the network utilization. This was a crowded area, defining (and trying to solve) important problems with (little) justification from the system perspective: was it reasonable to assume content owned by a particular user to reside solely on remote machines? Hence, we decided to focus on a specific instance of the storage problem, namely data backup, which offered a rich set of new challenges, while at the same time helped in defining a set of reasonable working assumptions.

[1]Churn is due to peers/users joining and leaving the system at any time.

The work we have thus developed has two main components. The first, covers our effort towards the definition of models of distributed storage systems deployed as P2P applications; our approach relies on non-cooperative game theory, which we use to understand and design incentive mechanisms aiming at mitigating the intermittent on-line behavior of users. The second, focuses on the design of algorithms and systems to offer efficient (both in terms of resource utilization and operational costs) and fast on-line storage service, whereby storage and bandwidth resources of end-hosts can be complemented by those available at servers, which are inherently more stable. We can summarize the main contributions of this set of work as follows.

We begin with a new formulation of the well-known problem of stable matching: this kind of problems can be modeled by a set of individuals willing to match with a set of other individuals of a different population (a noteworthy sub-class of stable matching is that of stable marriage problems), based on a preference list (a ranking of individuals that are candidates for the match). In our formulation, which is based on non-cooperative game theory, we model selfish individuals (that is, the users operating the storage application) that are ranked based on a heuristic aggregation of the “quantity" and “quality" of resources they offer to the storage application. The strategy available to the players is to gauge the amount of resources they dedicate to the system, which has the effect of modifying their ranking in the preference lists of other players.

Then, we focus on systems aspects of a specific variant of on-line storage applications, that is data backup. In this setting, we proceed with a new formulation of the problem of scheduling data transmission, which amounts to decide when and to which remote party to upload fractions of the data to be stored remotely. We show that the original combinatorial problem, which at a first glance appears to lack a polynomial-time solution, can be cast as a *network flow* problem, and we propose a solution based on well-known and efficient variants of Max-Flow algorithms. We also present a new approach to redundancy management, which is based on the realization that in a backup application: *i)* data durability is more important than data availability; *2)* the data access pattern is very particular, as backup data is written once (never updated) and possibly read rarely. As a consequence, our method to address the data redundancy problem uses a feedback mechanism to minimize (and adapt) the redundancy required to durably store data, and makes use of rate-less codes, which are a prerequisite to achieve adaptivity. Finally, we present a complete hybrid system design which leverages both peers and server resources, where the latter are seen as temporary buffers to boost data transfer performance.

Chapter 5 is dedicated to a new line of research of my group, that focus on data-intensive scalable computing systems (DISC). This domain represents a natural evolution of the research activities (and background) my team and myself developed in the past years: it involves massive amounts of data that need to be “moved", stored and processed.

The work I present in the Chapter has a strong “systems" component: in practice, it requires great zeal for building solid prototypes and repeatable experiments, in addition to produce original research in a very competitive domain. As a consequence, this work requires substantially more provisioning (both in terms of human resources and hardware material) than the previous topics addressed in this manuscript. In addition it should be noted that the research activity in this domain is largely dominated by the industry: many of the problems addressed in this domain stem from real-life issues that arise in production settings, which hinders the work of an academic research lab. To overcome the inherent limitation of working in an environment in which large-scale data management problems can only be imagined, my group and myself put a lot of effort in establishing collaborations with the industry, so as to be able to collect real-life data and work on relevant problems.

Although in this line of work we cover several aspects ranging from the very internal components of a DISC system, to high-level languages to overcome or extend the basic programming models of such systems, for the sake of brevity, Chapter 5 only covers in depth a small selection of our work[2]:

- Scheduling problems in DISC, which amount to study resource allocation. Production-level cluster traces, whereby DISC systems such as MapReduce (and its open source incarnation, Hadoop) are heavily used, reveal that typical workloads involve several tens of thousands of jobs submitted weekly, to process data for many different purposes. Recurrent analyses carried out by production jobs blend with ad-hoc queries on data and even experimental jobs: compute clusters are shared among users, because of a more efficient resource utilization and less redundancy. In this context, job scheduling plays a crucial role in determining the end-to-end performance (which we measure formally with the sojourn time of a job, amounting to the time a job spends in a queue waiting to be serviced and its execution time) of a DISC system. The line of research I develop in this manuscript, covers an attempt at bringing the benefits of a size-based approach to scheduling in a real system. This work has important implications: first, it shows that size-based scheduling (which is promoted by theoretical research, but largely disregarded in real systems) can be used in practice, and that it brings advantages going beyond an improved end-to-end performance. In addition, it represents a basic building block that my group and I will exploit to cover an open problem in DISC and traditional database systems: concurrent work sharing (aiming at efficient resource utilization by avoiding redundant computation), in which scheduling also plays a crucial role.

- The definition of new design patterns for the functional-style MapReduce programming model and their inclusion in new high-level abstractions that aim at simplifying the design of scalable algorithms. In this line of work, we tackle the

[2]A prominent example of a research line that is left outside of this manuscript is that of one of the PhD students I supervise, Xiaolan Sha, who graduated during the preparation of this manuscript. Her work deals with the design of scalable machine learning techniques to study time series data, predict trends and build recommender systems.

problem of optimizing the compilation of high-level languages into MapReduce primitives, and focus – to begin with – on an omnipresent primitive in on-line analytic processing (OLAP) system, that operates on hierarchical data and provides layered aggregates. Working on the compiler, in conjunction with the scheduler as defined above, allows the implementation of concurrent and non-concurrent work sharing techniques. The evolution of this line of work touches upon two main subjects: brining cost-based optimization techniques to DISC systems, similarly to what have been done in the traditional database system domain, to complement or eventually replace rule-based optimization; studying models of the MapReduce programming paradigm to better understand the trade-offs (and costs) related to the implementation of parallel algorithms.

Chapter 6 is dedicated to my teaching activities, including past and current courses I coordinate and teach, and with a Section dedicated to my plans for the evolution of my courses in the coming years.

Finally, **Chapter 7** summarizes the main contributions I achieved and outlines the research agenda of my group for the next years.

Chapter 3

Content Replication in Mobile Wireless Networks

Contents

Introduction

Academic and industrial research in the networking field is pursuing the idea that networks should provide access to contents, rather than to hosts. This goal has been extended to wireless networks as well, as witnessed by the tremendous growth of services and applications offered to users equipped with recent mobile terminals. Now, the inexorable consequence of a steady increase in data traffic exerted by mobile devices fetching content from the Internet is a drainage of network resources of mobile operators [4, 7]. As such, the problem of content distribution for mobile devices – and solutions to avoid or at least mitigate network congestion – has gained a lot of momentum.

The literature in the domain of content distribution over the Internet in general, and for mobile wireless networks in particular, is abundant. For an overview of related work, we defer the reader to Section 3.1. In general, prior works focus on simple network scenarios, in which mobile nodes interact through an ad-hoc wireless network, and the content distribution process is confined within that network:

a subset of the nodes are assumed to be initialized with an immutable copy of the content, stored locally, and are responsible for disseminating it to all other nodes. Despite elegant solutions for this particular setting have flourished, the underlying network model limits the extent to which access congestion – that relates to delayed access to content and poor download performance – can be studied. Indeed, depending on how sensitive to delays users are, there is nothing to prevent them to exhaust the resources of a cellular network or, alternatively, to overcrowd the wireless, device-to-device network, especially in case of "flash-crowds".[1]

The work presented in this Chapter departs from the existing literature because it uses a more realistic – and thus more elaborate – network model: the content distribution process begins in the wired Internet, in which a server (or a content distribution network) holds the original content and mobile users attempt at downloading it, whether directly – using a cellular network – or indirectly, using device-to-device communications.

First, we present a promising approach to address the problem of content distribution in the realistic setting described above. Our work suggests that content distribution can be cast as a replication problem: precisely, we formulate a variant the well known *facility location problem* and design a distributed algorithm capable of approximating an optimal solution, despite network and content demand dynamics.

Originally, our solution to the content distribution problem holds in a cooperative environment, that is where nodes abide to the algorithmic specification of our replication mechanism. With the intent of relaxing such a fundamental assumption, we extend our initial model to study the impact of a **non-cooperative** setup. Using non-cooperative game theory, we cast the replication problem as an *anti-coordination game*. In particular, our formulation accounts for content access delays caused by congestion (modeled as a contention problem due to concurrent access to network resources). In analogy to a "down-to-earth" setting, our model is akin to a well-known formulation of the car-pooling problem: users have to decide whether to use their own car or pool together and share a car to go to some destination. Using a private car implies a direct route to the destination, as opposed to a multi-hop route required to collect people prior to arriving at destination. However, if all users decide to use their own car, there will be congestion, while car-pooling lanes will be essentially free. The tension that exists between a direct route (and hence potentially low delays) and road congestion, are the essence of the game.

This line of work allows us to arrive at the conclusion that a randomized strategy – such as the one we use in the cooperative setting outlined above – represents an equilibrium strategy also in the non-cooperative scenario. A number of challenging open problems conclude this work.

We then conclude the Chapter by focusing on user mobility, and specifically study human mobility with an original approach: we use traces collected from a

[1]Flash-crowd indicates a phenomenon by which a sudden raise in popularity of a content triggers simultaneous attempts at downloading it.

popular[2] networked virtual environment – namely Second Life – and show that "avatars" exhibit mobility traits that closely match those of real humans. As a consequence, this work shows that the performance analysis of any mechanism involving mobile terminals operated by humans can be assessed by complementing (as we did in our work) mobility models with realistic traces extracted from virtual worlds.

3.1 Distributed Replication Algorithm

Traditionally, the problem of content distribution has been addressed with simple, widely used techniques including gossiping and epidemic dissemination [59], where content is forwarded to a randomly selected subset of neighboring nodes. Other viable – albeit more complex – approaches are represented by probabilistic quorum systems for information dissemination and sharing [52, 77]. In particular, in [52] the authors propose a mechanism akin to random walks to build quorums of users that would then exchange information. Node grouping has also been exploited in [58], where groups with stable links are used to cooperatively store and share information. Alternatively, some works attempted at modifying widely used peer-to-peer applications designed for the Internet (such as BitTorrent [41]) to operate on mobile wireless networks [81, 93].

In this Section, we overview a different, promising approach to solve the problem: *content replication*, that is used to create copies of information content at user devices so as to exploit device-to-device communication for content delivery. This approach has been shown to be effective especially in wireless networks with medium-high node density, where *access congestion* is the main culprit for the poor performance of content delivery (see, e.g., [44] for a survey on the topic).

In this line of work, we consider a mobile network and initially explore the concept of content replication in a **cooperative** environment: nodes can fetch content from the Internet using a cellular network, store it, and possibly serve other users through device-to-device communication (e.g., IEEE 802.11) [34]. Our scenario accommodates the possibility for content to exhibit variegate popularity patterns, as well as to be updated upon expiration of a validity-time tag, so as to maintain consistency with copies stored by servers in the Internet.

The scenario we target introduces several problems related to content replication. Our endeavor is to build upon the theoretic works that have flourished in the Location Theory literature and address the *joint problem* of content replication and placement, with the goal of designing a lightweight, distributed mechanism.

[2]This work was done in 2008, when Second Life was at the cusp of a rapid expansion in terms of user activity, and adoption. Today, Second Life has lost traction in favor of more specialized virtual worlds. Yet, our methodology has proven the point that measuring human mobility does not necessarily require costly experiments and "real" humans.

3.1.1 Problem Statement

First, we formalize the system model that applies to our application scenario. Based on this model, we inherit the problem of replication typical of the wired Internet and we discuss the new challenges introduced by the dynamic nature of wireless networks.

System model: We investigate a scenario including mobile users (*i.e.,* nodes), equipped with devices offering 3G/4G Internet connectivity as well as device-to-device communication capabilities (e.g., IEEE 802.11).

We denote the set of mobile nodes by $\mathcal{V}$, with $V = |\mathcal{V}|$, and we consider that users may be interested in a set of "abstract" information items, $\mathcal{I}$ ($|\mathcal{I}| = I$). Each item $i \in \mathcal{I}$, of size $s(i)$, is tagged with a validity time and originally hosted on a server in the Internet. We define the content popularity of a generic item i, $\pi(i)$, as the fraction of users/nodes interested in such an item. Thus, we have $0 \leq \pi(i) \leq 1$, with $\pi(i) = 1$ when all nodes in the system are interested in content i.

In this line of work we focus on a **cooperative environment**: a node $j \in \mathcal{V}$ interested in an information item i first tries to retrieve (which involves content lookup and download) it from other devices. If its search fails, the node downloads a *fresh content replica* from the Internet server and temporarily stores it for a period of time τ_j, termed *storage time.* For simplicity of presentation, we assume $\tau_j = \tau$, $\forall j \in \mathcal{V}$. During the storage period, j serves the content to other nodes upon receiving a request for it and, possibly, downloads from the Internet server a fresh copy of the content if its validity time has expired. We refer to the nodes hosting an information copy at a given time instant as *replica nodes.* We denote the set of nodes storing a copy of item i at time t by $\mathcal{R}_i(t)$, and define $\mathcal{R}(t) = \cup_{i \in \mathcal{I}} \mathcal{R}_i(t)$, with $R = |\mathcal{R}|$. Also, we associate to each replica node j a capacity value c_j, which, as we shall see later, relates to the capability of the node to serve content requests.

A node, which is interested in a generic information item i and does not store any copy of it, issues queries for such an item at a rate λ. Replica nodes, receiving a query for an information item they currently store, will reply with a message including the requested content.

In the following we model the network topology at a given time instant t through a graph $G(t) = (\mathcal{V}, \mathcal{E}(t))$, whose set of vertices coincides with the set of nodes $\mathcal{V}$ and the set of edges $\mathcal{E}(t)$ represents the set of links existing between the network nodes at time t.

The Problem: Traditionally, content replication has been studied through the lenses of Location Theory, by considering replicas to be created in the network as facilities to open. As the first step to understand the problem under study, we restrict our attention to a simplified network setting and revisit a centralized approach for facility location problems. To simplify mathematical notation, we assume static nodes and constant demand, hence we drop the time dependency from our notation; we also let all users be interested in every content i ($i = 1, \ldots, I$).

Given such a simplified scenario, we formulate content replication as a **capacitated facility location problem** where the set of replica nodes $\mathcal{R} = \cup_i \mathcal{R}_i$ corresponds to the set of facilities that are required to be opened, nodes requesting content are referred to as clients and items correspond to the commodities that are available at each facility. We model the capacity of a replica node as the number of clients that a facility can serve. The goal is to identify the *subset* of facilities that, at a given time instant, can serve the clients so as to minimize some global cost function while satisfying the facility capacity constraints. Note that, in our scenario, both clients and facilities lay on the same network graph $G = (\mathcal{V}, \mathcal{E})$. The problem can be defined as follows:

Definition 1 *Given the set $\mathcal{V}$ of nodes with pair-wise distance function d and the cost f_j of opening a facility at $j \in \mathcal{V}$, select a subset of nodes as facilities, $\mathcal{R} \subseteq \mathcal{V}$, so as to minimize the joint cost $C(\mathcal{V}, f)$ of opening the facilities and serving the demand while ensuring that each facility j can only serve at most c_j clients. Let $C(\mathcal{V}, f)$ be:*

$$C(\mathcal{V}, f) = \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{R}_i} f_j(i) + \sum_{i \in \mathcal{I}} \sum_{h \in \mathcal{V}} d(h, m_h(i)) \tag{3.1}$$

where $f_j(i)$ is the cost to open a facility for commodity i, $\mathcal{R}_i \subseteq \mathcal{V}$ is the subset of nodes acting as facilities for commodity i, $m_h(i) \in \mathcal{R}_i$ is the facility holding item i that is the closest[3] *to h, and the number $u_j(i)$ of clients requesting any content i attached to facility $j \in \mathcal{R}_i$, i.e., $u_j(i) = |\{h \in \mathcal{V} \text{ s.t. } m_h(i) = j\}|$, is such that $\sum_{i \in \mathcal{I}} u_j(i) \leq c_j$.*

Note that our problem formulation is more complex than the traditional one, where the intersection between the sets of facilities and clients is null. Indeed, since in our settings any vertex of the graph G can host a facility (*i.e.,* be a replica node for an item) or be a client (*i.e.,* request an item that does not currently own), a vertex can assume both roles. Moreover, in the location theory literature, two copies of the same facility can be opened at the same location, in order to increase the capacity of a site. Instead, in our work a vertex of the graph can host only one copy of the same facility, as it is reasonable that a node stores only one copy of the same item.

Finding approximate solutions to the problem of multi-commodity capacitated facility locations, even in its (simpler) traditional formulation, is an open issue and little is known concerning heuristics that can be effectively implemented in practice. In our work, we take the following approach: a solution to the multi-commodity problem is built from the union of the solutions to individual single-commodity facility location problems. We transform the formulation from multi-commodity to single-commodity by solving the above problem for each item i ($i = 1, \ldots, I$) separately[4]. Then, we denote the subset of commodities hosted at node j by $\mathcal{I}_j$ and its cardinality by I_j, and we adopt two different techniques to verify the capacity constraints:

[3] As distance function, we take the Euclidean distance between the nodes.

[4] A single-commodity facility location problems reduces to the k-median problem when the number of facilities to be opened, k, is given.

1. each opened facility (replica node) has a capacity that is allocated to each commodity individually: this translates into having a separate budget allocated to each commodity (item). The capacity constraints can be written as $u_j(i) \leq c_j/I_j$, $\forall i \in \mathcal{I}_j$, where we equally split the budget c_j available to facility j over all the commodities it hosts. In the following, we name such a technique *split capacity budget*;

2. we consider that the capacity of a facility is shared among the commodities it currently hosts, i.e., each replica node allocates a preset budget that is used to serve the requests by other nodes. We write the capacity constraints for this case as: $\sum_{i \in \mathcal{I}_j} u_j(i) \leq c_j$, and we refer to such a technique as *shared capacity budget*.

To solve such a problem, we resort to the local search heuristic detailed in [13], which finds a solution to the capacitated, single-commodity location problem that is one of the best known approximations to optimal replication and placement. Hereinafter we term such a heuristic *centralized* facility location (CFL) algorithm because it can only be executed in a centralized, synchronous environment. We consider the CFL algorithm to be a baseline against which we compare the results obtained by our distributed approach.

Note that existing distributed approximation algorithms that tackle facility location problems either require global (or extended) knowledge of the network [13] or are unpractical [82]. Therefore, in the next section we propose a new approach that only requires local knowledge, which is acquired with simple measurements, and adapts to the system dynamics. In addition, our scheme provides load-balancing; it follows that, even in a static scenario, our distributed algorithm would not converge to a static configuration in which a fixed set of nodes is selected to host content replicas. As such, the traditional methods that are used in the literature to study the convergence properties and the locality gap of local search algorithms cannot be directly applied, which is the main reason for us to take an experimental perspective and validate our work through simulation.

3.1.2 Distributed algorithm

Armed with the insights on the problem formulation discussed above, our mechanism mimics a local search procedure, by allowing replica nodes to execute one of the following three operations on the content: (1) handover, (2) replicate or (3) drop. However, unlike the traditional local search procedures, in our mechanism the three operations yield the solution to the content replication problem iteratively, albeit *asynchronously*. Furthermore, in our network system, replicate and handover are constrained operations: only vertexes that are connected by an edge to the current vertex hosting a content replica can be selected as possible replica locations. Thus, our operations are *local* and replicas can only move by one hop at the time in the underlying network graph.

In the following we describe our mechanism in terms of two objectives: content *replication* and *placement*. Indeed, the handover operation amounts to solving the optimal placement of content replicas, whose number is determined through the replicate and drop operations. For simplicity, we consider again that all users are interested in every content i ($i = 1, \dots, I$) and we fix the time instant, hence we drop the time dependency from our notation.

Content replication: Let us define the workload of the generic replica node j for content i, $w_j(i)$, as the number of requests for content i served by j during its storage time. Also, recall that we introduced the value c_j as the capacity of node j and we provided a definition that suited the simplified, static scenario described in Section 3.1.1. We now adapt the definition of c_j to the dynamic scenario at hand, as the reference volume of data that replica node j is willing to provide during the time it acts as a replica node, *i.e.,* in a storage time τ. Then, with reference to Eq. (3.1), we denote by $f_j = \sum_{i \in \mathcal{I}_j} f_j(i)$ the cost that a node j must bear while acting as a facility for any content.

Given the load balancing property we wish to achieve across all replica nodes and the capacity constraints, the total workload for replica node j should equal c_j. Thus, we write f_j as:

$$f_j = c_j - \sum_{i \in \mathcal{I}_j} s(i) w_j(i) \tag{3.2}$$

In other words, we let the cost associated with replica node j grow with the gap between the workload experienced by j and its capacity c_j.

Then, during storage time τ, the generic replica node $j \in \mathcal{R}$ measures the number of queries it serves, i.e., $w_j(i)\ \forall i \in \mathcal{I}_j$. When its storage time expires, the replica node j computes f_j and takes the following decisions: if $f_j > \varepsilon$ the content is *dropped*, if $f_j < -\varepsilon$ the content is *replicated*, otherwise the handover operation is executed (see below). Here, ε is a tolerance value to avoid replication/drop decisions in case of small changes in the node workload.

The rationale of our mechanism is the following. If $f_j < -\varepsilon$, replica node j presumes that the current number of content replicas in the area is insufficient to satisfy the current content request workload: hence, the node replicates the content and hands the copies over to two of its neighbors (one each), following the placement mechanism described below. The two selected neighbors will act as replica nodes for the subsequent storage time. Instead, if $f_j > \varepsilon$, node j estimates that the current number of replicas can satisfy a workload that exceeds the total demand: thus, it drops the content copy. Finally, if the experienced workload is (about) the same as the reference value, replica node j selects one of its neighbors to which to hand over the current copy, again according to the mechanism detailed next.

Replica placement: As noted in Section 3.1, given the graph representing the network topology at a fixed time instant, the placement of $R{=}k$ replicas can be cast as a k-median problem. By applying the approximation algorithm in [13], in [27] we observed that the solution of such a problem for different instances of the topology graph yields replica placements that are instances of a random variable uniformly distributed over the graph. Thus, in a dynamic environment our target is to design a distributed, lightweight solution that closely approximates a uniform distribution of the replicas over the network nodes while ensuring load balancing among them. To this end, we leverage some properties of random walks and devise a mechanism, called *Random-Walk Diffusion (RWD)*, that drives the "movement" of replicas over the network.

According to RWD, at the end of its storage time τ, a replica node j randomly selects another node l to store the content for the following storage period, with probability $p_{j,l} = \frac{1}{d_j}$ if l is a neighbor of j, and 0 otherwise, where d_j is the current number of neighbors of node j. In this way, each replica performs a random walk over the network, by moving from one node to another at each time step τ. Thus, we can apply the result stating that in a connected, non-bipartite graph, the probability of being at a particular node j converges with time to $d_j/(2|\mathcal{E}|)$ [75]. In other words, if the network topology can be modeled by a regular graph[5] with the above characteristics, the distribution of replicas moving according to a random walk converges to a stationary distribution, which is uniform over the nodes. In general, real-world networks yield non-regular graphs. However, when V nodes are uniformly deployed over the network area and have the same radio range, the node degree likely has a binomial distribution with parameters $(V-1)$ and p, with p being the probability that a link exists between any two nodes [60].

For practical values of p and V in the scenarios under study, we verified that the node degree distribution is indeed binomial with low variance, *i.e.,* all nodes have similar degree. It follows that a random walk provides an acceptable uniform sampling of the network nodes, hence the replica placement distribution well approximates the uniform distribution.

A similar result can be obtained also for clustered network topologies, where each cluster core results to be an expander graph [16]. In this case, a uniform replica placement over the nodes can be achieved within each of the network clusters, thus ensuring the desired placement in all areas where the user demand is not negligible.

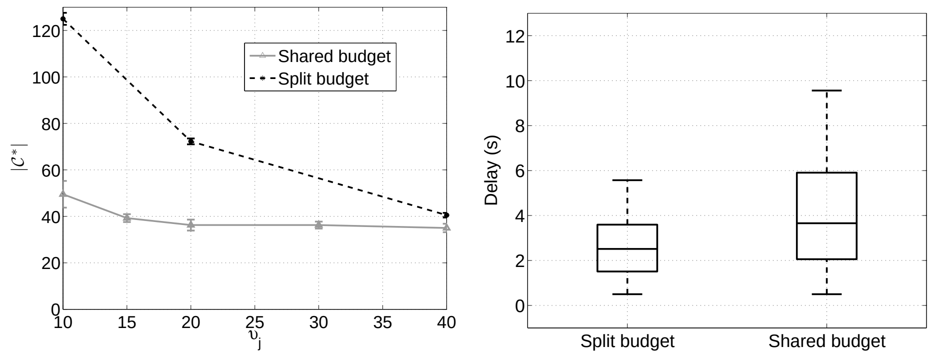
Finally, we stress that the presence of R replicas in the network corresponds to R parallel random walks. This reduces by almost a factor R the expected time to sample all nodes in the network, which is closely related to the time needed to approximate the stationary distribution by a constant factor [73]. It follows that, given a generic initial distribution of the replicas in the network, the higher the R, the more quickly the replica placement approximates a uniform distribution.

3.1.3 Experimental results

This section overviews a selection of the experimental results obtained with a simulation-based approach. A comprehensive set of results, including various kinds of mobility models, and the simulation setup is available as a technical report [68].

We organize the main results of our work in several sections that cover the parameter space we studied. To benchmark our distributed mechanism against the

[5]A graph is regular if each of its vertices has the same number of neighbors.



(a) Number of replicas (b) Delay

Figure 3.1: Numerical solutions of the optimization problems in terms of number of replicas (a) and query solving delay (b). In (b), we show the 5%, 25%, 50%, 75% and 95% percentiles

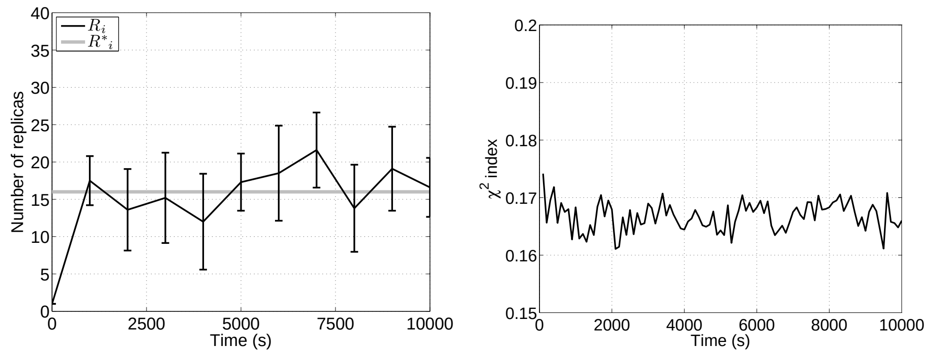
centralized approach discussed in the introductory part of this work, we implement the CFL algorithm as follows. Given the network time evolution, we take a snapshot of the network topology every τ s. For every snapshot, we solve I separate single-commodity problems derived from Eq. (3.1), under both split and shared capacity budgets. To do so, we set $f_j(i) = c_j/I_j - u_j(i)$ and $f_j = c_j - \sum_{i\in\mathcal{I}_j} u_j(i)$ in the case of split and shared capacity budget respectively, with $u_j(i) = s(i)w_j(i)$.

Benchmarking the replication scheme: First, we study the impact of the allocation of the node capacity budget. We take a numerical approach and focus on the CFL algorithm: our objective here is to determine the implications of *split* or *shared* capacity allocations.

The optimal number of replicas per information item, denoted by C_i^*, is obtained by numerically solving the optimization problem in Def. 1, in both its split and shared capacity budget versions, and is shown in Fig. 3.1(a). The plot shows that, as higher budgets allow replica nodes to satisfy larger amounts of requests, increasing c_j reduces the need for replication thus leading to a lower number of replicas in the network. Using a common budget for all items (i.e., shared capacity budget), forces replications only when the total workload for all items exceeds the budget. Conversely, optimization with split capacity budget uses separate budgets for each content and, thus, results in more frequent violations of such constraints.

Intuitively, more replicas should imply higher chances for queries to be satisfied through device-to-device communications. In Fig. 3.1(b) we show the most important percentiles of content access delay. Contrary to the intuition, our results indicate that the advantage granted by a high number of replicas under the split capacity is quite negligible: indeed, the lower number of replicas deployed by the shared capacity allocation suffices to satisfy most of the requests generated by nodes in the ad hoc network.

In summary, these first experiments pinpoint that the replication mechanism



(a) Number of replicas (b) χ^2 index

Figure 3.2: Numerical solutions of the optimization problems, and comparison against our replication scheme: temporal evolution of the number of replicas (a), and of the χ^2 index (b)

with shared capacity constraints is a suitable approach. Beside experimental results, there are also practical reasons to opt for shared capacity constraints. Indeed, in the split capacity case, a budget has to be assigned to each item currently stored by a replica node, which is a quantity that may vary over time. As a consequence, content replicas may not be suitably handled if the remaining capacity available to a node is not appropriately re-distributed. Furthermore, it would be unfeasible to ask a user to select a service budget to allocate to every possible item she will ever replicate. In the following we will therefore focus on the *shared* capacity budget only.

Next, we simulate our distributed replication scheme and compare it to the CFL algorithm. As shown in Fig. 3.2(a), our distributed algorithm approximates well the results obtained by solving the optimization problems in a centralized setting: indeed, the number of replicas R_i generated by our scheme is very close to the optimal value R_i^*. We then study the similarity between the replica placement achieved by our technique and that obtained with the CFL algorithm. To do so, we employ the well-known χ^2 goodness-of-fit test on the inter-distance between content replicas As depicted in Fig. 3.2(b), the χ^2 error we obtain is well below the value (namely, 23.685) needed to accept the null hypothesis that the two distributions are the same at a 95% confidence level.

Comparison to other approaches: We now consider information items to be associated to different popularity levels, and compare the performance of our replication scheme with that of the square-root replication strategy [42]. According to such a strategy, the allocation percentage for a content i is proportional to the square root of the total demand per second for that content. In [42], it has been proved that square-root replication is optimal in terms of number of solved queries.

Fig. 3.3 shows the fraction of the total number of replicas of item i, versus the associated query rate $\pi(i)V\lambda$, for $I = 4$ and c_j={5, 15, 40} Mbytes. The plot compares our scheme with: (i) the square-root strategy, (ii) a uniform strategy, which allocates

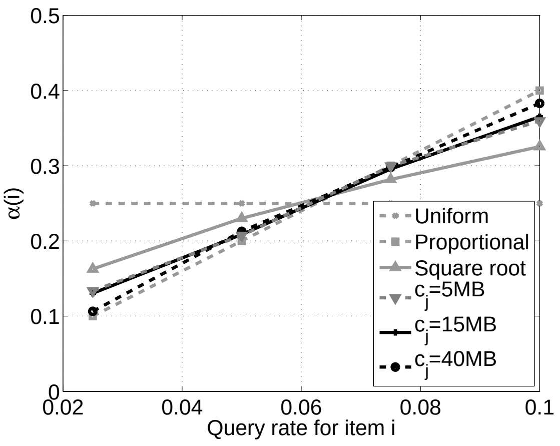


Figure 3.3: Fraction of replicas for each of the four items, in comparison with uniform, proportional and square-root allocation

the same number of replicas per item, and (iii) a proportional strategy, where the number of replicas is proportional to the content popularity. Our solution achieves an allocation in between the square-root and proportional distributions, while it is far from that obtained under the uniform strategy. This suggests that our replication mechanism well approximates the optimal replication strategy. In particular, when c_j is higher, i.e., replica nodes are more generous in reserving resources to serve requests, the allocation tends to follow a proportional distribution. Conversely, in presence of lower values of c_j the allocation better fits the square-root rule.

Since our replication scheme roughly achieves the result obtained by a square-root allocation, it is reasonable to wonder why a different approach to content replication is required. First of all, we have different objectives than that of [42]: load-balancing, for example, requires an additional layer to complement the square root allocation scheme, which instead we achieve as part of our design. Furthermore, the distributed version of the replication algorithms proposed in [42] has some limitations that render them less suitable to be deployed in a mobile, wireless environment. The simple path replication scheme catering to low storage requirements, just like our scheme, substantially over/undershoots the optimal number of replicas. The other approaches discussed in [42] are better at converging to an optimal number of replicas but require the bookkeeping of large amounts of information. Finally, the design and the evaluation of such algorithms in [42] are performed in a static wired environment and do not take into account the dynamics typical of a mobile network, such as that we consider.

To complete the comparative study, we also implement a simple caching scheme and compare its performance to that obtained with our distributed replication mechanism. In summary (details can be found in [68]), an approach based on content replication achieves smaller content access delays – that is, the time required to fetch the content, whether from the Internet using the cellular network, or from other nodes using device-to-device communications – and results in fewer downloads from the (costly) cellular network.

In conclusion, our scheme clearly emerges as a simple, efficient and performing alternative to traditional mechanisms: by controlling the replicas number and placement, it appears to be suitable especially when content popularity is not 100%. Further experiments, omitted here for the sake of brevity, also show that our algorithm achieves load balancing in a variety of settings, including multiple contents with variable popularity, and scales well with node density and number of contents, whereby the effects of scale are negligible both on load balancing and in access delay.

3.2 Selfish Replication

We now address the problem of content replication in a hybrid wireless network under the assumption that nodes are not cooperative, *i.e.,* they do not abide to a specific algorithm when taking decisions to replicate content. Similarly to what discussed in Section 3.1, we assume a "flash-crowd" scenario, in which users discover a new content and wish to access it concurrently. As a consequence, *access congestion* determines to a large extent the download performance, for both the cellular and the ad hoc network. To simplify the problem formulation, here we consider nodes to be interested in a single information object.

The problem of replication (and caching) has received a lot of attention in the past due to its importance in enhancing performance, availability and reliability of content access in wireless systems. However, this problem has been addressed often under the assumption that nodes would cooperate by following a strategy that aims at optimizing the system performance, regardless of the costs incurred by each individual node. Our goal is similar to the one described in the seminal work [39], in that we build a model where nodes are selfish, *i.e.,* they choose whether to replicate or not some content so as to minimize their own cost. Our work differs from [39] in how content demand is modeled.

3.2.1 Problem statement

Let $\mathcal{I}$ be a set of nodes uniformly deployed on area $\mathcal{A} = \pi R^2$. We consider a single base station covering the network area, and we denote by $r < R$ the radio range that nodes use for device-to-device communications. Also, let the information object be of size equal to L bytes; the object requires f updates per second from the origin server (in order to obtain a fresh copy) and each update implies the download of U bytes.

We now formulate the problem as a (simultaneous move) replication game: $\mathcal{I}$ is the set of players, with $|\mathcal{I}| = I$, and $\mathcal{S}_i = \{1, 0\}$ is the set of all possible strategies for player $i \in \mathcal{I}$.

Additionally, let $s_i \in \mathcal{S}_i$ be the strategy of player i, where $s_i = 1 \vee 0$. The array $s = \{s_1, s_2, ..., s_i, ..., s_I\}$ is a strategy profile of the game.

Furthermore, let $\mathcal{C} \subseteq \mathcal{I}$ be the set of players whose strategy is to access the object from the origin server and cache it, that is $s_i = 1, \forall i \in \mathcal{C}$, and $\mathcal{N} \subseteq \mathcal{I} = \mathcal{I} \setminus \mathcal{C}$ the set of players whose strategy is to access a replica object served by another player, that is $s_i = 0, \forall i \in \mathcal{N}$. Finally, let $|\mathcal{C}| = x$ and $|\mathcal{N}| = I - x$.

In this game we need to impose $\mathcal{C} \neq \emptyset$: at least one player has to replicate the object for otherwise content would not be available to any player. Given a strategy profile s, the cost incurred by player i is defined as:

$$C_i(s) = \beta_i \mathbb{1}_{s_i=1} + \gamma_i \mathbb{1}_{s_i=0}$$

where β_i and γ_i are the air time costs if i obtains the content through the cellular network and via device-to-device communication, respectively, and $\mathbb{1}_{s_i}$ is the indicator function. In this work we focus on access costs, neglecting the energy cost that a node caching the object experiences when serving other nodes.

We now define precisely the two terms β_i and γ_i. To this end, let us introduce the following quantities: R_c and R_h are the bit rates offered, respectively, by the cellular and the ad hoc network; T_c is the time for which node $i \in \mathcal{C}$ caches the object; h is the *average* number of hops required to access the closest cached object, assuming a uniform distribution of nodes on $\mathcal{A}$ and a *uniform distribution of caches.* Formally, $h = \sqrt{\frac{\mathcal{A}}{x\pi r^2}} = \frac{R}{r}\frac{1}{\sqrt{x}}$.

With these definitions at hand, we can now focus on the two cost terms, β_i and γ_i:

$$\beta_i = |\mathcal{C}|\frac{L + T_c f U}{R_c} = |\mathcal{C}|k_1 \qquad \gamma_i = \frac{|\mathcal{N}|}{\sqrt{|\mathcal{C}|}}k_2 \ , \ k_2 = \frac{R}{r}\frac{L}{R_h}$$

β_i distinguishes "installation" and "maintenance" costs and models the congestion incurred by nodes trying to access the information object at the same time: the bit rate R_c is inversely proportional to the number of concurrent users accessing a *single* 3G base station [105]. Similarly, γ_i represents the air time consumed to access the current version of the cached object and models the congestion cost created by simultaneous access of nodes operating in ad hoc mode[6].

3.2.2 Results

First, we focus on the analysis of the game by computing the so-called **social optimum**, which is defined, for a given strategy profile, as the total cost incurred by all players, namely:

$$C(s) = \sum_{i\in\mathcal{C}} \beta_i + \sum_{i\in\mathcal{N}} \gamma_i \qquad \rightarrow \qquad C(x) = x^2 k_1 + \frac{(I-x)^2}{\sqrt{x}}k_2$$

The minimum social cost is $\hat{C}(s) = \min_s C(s)$. Our results are quite intuitive: as the communication capability of nodes increase – for example, when the transmission radio range increases, or when the data rate for device-to-device

[6]Our congestion model is more conservative than the capacity scaling law defined in [57].

communication increases – the number of replicas in the socially-optimal solution decreases sharply. Instead, when the area in which nodes are deployed increases (but the number of nodes is constant), the number of content replicas also has to increase. The interested reader may refer to [80] for more details on the solutions we obtain by solving the optimization problem.

Next, we focus on a simple **two-player game**, derive its equilibrium points and compare their efficiency to the social optimum. The two-player version of the replication game involves two players whose strategy set is $S_i = \{1, 0\}$ as defined above. For clarity, we label $s_i = 1 \to C$ and $s_i = 0 \to N$. Table 3.1 illustrates the normal form game[7].

	$p_2(C)$	$p_2(N)$
$p_1(C)$	$\left(\frac{1}{4k_1}, \frac{1}{4k_1}\right)$	$\left(\frac{1}{k_1}, \frac{1}{k_2}\right)$
$p_1(N)$	$\left(\frac{1}{k_2}, \frac{1}{k_1}\right)$	$(0, 0)$

Table 3.1: Payoff matrix, π_{ij}. $p_i(x)$ indicates player i choosing strategy x.

Clearly, strategy N is *strictly dominated* by strategy C if and only if $4k_1 < k_2$: in this case, we would have only one Nash Equilibrium (NE), which is (C, C). Instead[8], when $4k_1 > k_2$, we face an *anti-coordination game*, in which players randomize their strategies. Indeed, there are two conflicting (in terms of payoffs) NE points, *i.e.*, the (N, C) and (C, N) strategy profiles. It is well known that mixed-strategies profiles and expected payoffs π_i can be derived as follows. Suppose player 2 chooses C with probability α, then $\mathbb{E}[\pi_1(C, \alpha)] = \frac{4-3\alpha}{4k_1}$ and $\mathbb{E}[\pi_1(N, \alpha)] = \frac{\alpha}{k_2}$. Hence, $\alpha = \frac{4k_2}{4k_1+3k_2}$. Due to the symmetry of the game, player 1 chooses C with probability $\beta = \alpha$. Considering the joint mixing probabilities, the expected payoff for both players is $\mathbb{E}[\pi_i^*] = \frac{4}{4k_1+3k_2} \, \forall i \in 1, 2$.

It is worth noting that in this anti-coordination game the mixed strategy NE is *inefficient*. Indeed, when players can correlate their strategies based on the result of an observable randomizing device (*i.e.*, a *correlated equilibrium*), the expected payoff is $\mathbb{E}[\hat{\pi}_i] = \frac{k_1+k_2}{2k_1k_2} \, \forall i \in 1, 2$. We observe that $\mathbb{E}[\hat{\pi}_i]$, which corresponds to the social optimum, is *strictly larger* than $\mathbb{E}[\pi_i^*]$. This clearly suggests that some correlation among the nodes' actions should be introduced in order to improve system performance.

3.2.3 Discussion and open problems

The results above can be extended to an n-player setting, which is an extension to this work we didn't pursue further. Instead, our concern is to pur into a practical

[7]When no player replicates the object, access costs are infinite.

[8]This is the case that happens in practice, e.g., with the values of the system parameters used to compute the minimum cost.

perspective our theoretic findings. We do so by defining two problems that are left open for the research community to address.

- On the one hand, in the n-player replication game, a player can compute her best response if the current number x of caches in the network is known. Since in practice global knowledge cannot be assumed, it is important to study how far from efficiency the system settles when nodes compute an estimate of x. Such an estimate can be obtained either through random sampling techniques based on gossiping, or by exploiting *local measurements* of the number of queries received by each node caching the content (as we did in the cooperative scenario described in detail in Section 3.1). A further open question is to determine how sensitive to estimation errors the achieved equilibrium is.

- On the other hand, we observe that a correlated equilibrium is impractical in an *asynchronous setting*. To address this issue, it is common to allow communication between players through *signaling*, that replaces the external randomization device cited above and is used by a player to notify its strategy to others. The use of signaling however implies taking into account neighboring relations among players, as dictated by the underlying communication graph defined by the network topology.

3.3 Measuring Human Mobility

Characterizing the mobility of users has been the subject of several studies in a variety of domains, especially in that of wireless, mobile ad hoc networks (MANET) [26]. For example, the literature on MANET routing is rich in mobility *models* that have been designed, analyzed and used for simulation-based performance evaluation of ad hoc routing schemes [24, 61, 84]. Some of these models have also been heavily criticized in the literature [107].

In recent years, a new class of problems rised by the delay tolerant networking (DTN) paradigm has encouraged the study of *human mobility*. For example, [29, 30, 62] conducted several experiments mainly in confined areas and studied analytical models of human mobility with the goal of assessing the performance of message forwarding in DTNs. Experimental approaches such as the ones discussed above, rely on users volunteering to take part in such experiments. Users are equipped with a wireless device (for example a sensor device, a mobile phone, ...) running a custom software that records *temporal information* about their contacts. Individual measurements are collected, combined and parsed to obtain the temporal distribution of contact times. Real-life experiments are hindered by several factors, including logistics, software/hardware failures, scalability and are bound to specific events (e.g. conferences, public events).

Our contribution in this domain is a novel methodology to capture spatio-temporal dynamics of user mobility that overcomes most of the limitations of previous attempts: it is cheap, it requires no logistic organization, it is not bound to a specific wireless technology and can potentially scale up to a very large number of participants. Our measurement approach exploits the tremendous raise in popularity of Networked Virtual Environments (NVEs), wherein thousands of users connect daily to interact, play, do business and follow university courses just to name a few potential applications. Here we focus on the SecondLife (SL) "metaverse" [9] which has recently gained momentum in the on-line community.

Our primary goal is to perform a temporal, spatial and topological analysis of user interaction in SL. Prior works that attempted the difficult task of measuring and collecting traces of human mobility and contact opportunities are restricted by logistic constraints (number of participants to the experiments, duration of the experiments, failures of hardware devices, wireless technology used). In general, position information of mobile users is not available, thus a spatial analysis is difficult to achieve [30]. Some experiments with GPS-enabled devices have been done in the past [66, 91], but these experiments are limited to outdoor environments.

In the following, we outline the measurement tool we built to achieve our goals, which is essentially a *crawler*. Our crawler connects to SL and extracts position information of all users concurrently connected to a sub-space of the metaverse.

One striking evidence of our results is that they approximately fit real life data, raising the legitimate question whether measurements taken in a virtual environment present similar traits to those taken in a real setting. Our methodology allows performing large experiments at a very low cost and generate data that can be used in a variety of applications. Although our plan is to use them to perform trace-driven simulations of communication schemes in delay tolerant networks and their performance evaluation, our dataset can be used for social science and epidemiology studies not to mention their value for the design and evaluation of virtual world architectures.

3.3.1 Crawling Second Life

The task of monitoring user activity in the **whole** SL metaverse is very complex: in this work we focus on measurements made on a selected subspace of SL, that is called a land (or island). In the following we use the terminology *target land* to indicate the land we wish to monitor. Lands in SL can be private, public or conceived as sandboxes and different restrictions apply: for example private lands forbid the creation and the deployment of objects without prior authorization.

Mining data in a NVE can be approached from different angles. A possible approach is to create sensor objects so as to mimic the deployment of a sensor network which is used to capture user activity. However, there are several limitations intrinsic to this approach that hinder our ultimate goal, which is to collect a large data set of user mobility patterns. These limitations mostly come from inner design choices made by the developers of SL to protect from external attackers aiming at disrupting the system operation. An example of such attacks consists in indefinitely cloning of a simple object (such as a sphere). Due to the centralized nature of the SL architecture, which allocates a single physical machine to handle a land, its objects and its users, the simple procedure outlined above constitutes a very effective denial of service attack.

For this work, we adopt a different approach and build a custom SL client software (termed a *crawler*) using `libsecondlife` [8]. The crawler is able to monitor the position of **every** user using a specific feature of `libsecondlife` that enables the creation of simple maps of the target land. Measurement data is stored in a database that can be queried through an interactive web application. The crawler connects to the SL metaverse as a normal user, thus it is not confined by limitations imposed by private lands: any accessible land can be monitored in its totality; the maximum number of users that can be tracked is bounded only by the SL architecture (as of today, roughly concurrent 100 users per land); communication between the crawler and the database is not limited by SL.

During our experiments, we noted that introducing measurement probes in a NVE can cause unexpected effects that perturb the normal behavior of users and hence the measured user mobility patterns. Since our crawler is nothing but a stripped-down version of the legacy SL client and requires a valid login/password to connect to the metaverse, it is perceived in the SL space as an avatar, and as such may attract the attention of other users that try to interact with it: our initial experiments showed a steady convergence of user movements towards our crawler. To mitigate this perturbing effect we designed a crawler that mimics the behavior of a normal user: our crawler randomly moves over the target land and broadcasts chat messages chosen from a small set of predefined phrases.

3.3.1.1 Measurement methodology

Using the physical coordinates of users connected to a target land, we create snapshots of *radio networks*: given an arbitrary communication range r, a communication link exists two users v_i, v_j if their distance is less than r. In the following we use a temporal sequence of networks extracted from the traces we collected using our *crawler* and analyze contact opportunities between users, their spatial distribution and graph-theoretic properties of their communication network.

A precondition for being able to gather useful data is to select an appropriate target land and measurement parameters. Choosing an appropriate target land in the SL metaverse is not an easy task because a large number of lands host very few users and lands with a large population are usually built to distribute virtual money: all a user has to do is to sit and wait for a long enough time to earn money (for free). In this work, we manually selected and analyzed the following lands: *Apfel Land*, a german-speaking arena for newbies; *Dance Island*, a virtual discotheque; *Isle of View*, a land in which an event (St. Valentines) was organized. These lands have been chosen as they are representative of out-door (Apfel Land) and in-door (Dance Island) environments; the third land represents an example of SL events which supposedly attract many users. Next, we present results for 24 hours traces: while the analysis of longer traces yields analogous results to those presented here, long experiments are sometimes affected by instabilities of `libsecondlife` under a Linux environment and we decided to focus on a set of shorter but stable measurements. A summary of the traces we analyzed can be defined based on the total number of unique users and the average number of concurrently logged in users: Isle of View had 2656 unique visitors with an average of 65 concurrent users, Dance Island had 3347 unique users and 34 concurrent users in average and Apfel Land had 1568 users and 13 concurrent users in average.

We launched the crawler on the selected target lands and set the time granularity (intervals at which we take a snapshot of the users' positions) to $\tau = 10$ sec. We selected a communication range r to simulate users equipped with WiFi (802.11a at 54 Mbps) device, namely $r_w = 80$ meters. In this work we assume an *ideal wireless channel*: radio networks extracted from our traces neglect the presence of obstacles such as buildings and trees.

3.3.2 Results

We now discuss a selection of the main results of our measurement campaign for three selected target lands. Here we focus on the analysis of the statistical distribution of contact opportunities between users, which has been extensively covered by related work addressing experiments in the "real-world".

In this work, we used Maximum likelihood estimation (MLE) [40] for fitting our traces to well-known models of contact-time distributions. The three baseline models we used are summarized in Table 3.2. We applied MLE to analyze the distribution of contact times. The CCDF of the contact time CT can be best fit to an *exponential distribution*: when $r = r_b$ we have that $\lambda = \{0.010, 0.003, 0.008\}$ and when $r = r_w$ we have that $\lambda = \{0.007, 0.002, 0.004\}$ respectively for ApfelLand, Dance Island and Isle Of View. MLE applied to our empirical data on inter contact times indicates that the best fit is the the *power-law with cutoff distribution.* We observe that the CCDF of the inter contact time ICT has two phases: a first power-law phase and an exponential cut-off phase. The values of the coefficients of these distributions are: $\alpha = \{0.34, 0.47, 0.42\}$ and $\lambda = \{0.00049, 0.00041, 0.00046\}$ respectively for ApfelLand, Dance Island and Isle Of View, when $r = r_b$ and $\alpha = \{0.46, 0.44, 0.59\}$, $\lambda = \{0.00045, 0.00037, 0.00041\}$ when $r = r_w$.

Table 3.2: Definition of the power-law distribution and other reference statistical distributions we used for the MLE. For each distribution we give the basic functional form $f(x)$ and the appropriate normalization constant C.

name	$f(x)$	C
power-law	$x^{-\alpha}$	$(\alpha-1)x_{min}^{\alpha-1}$
power-law with cutoff	$x^{-\alpha}e^{-\lambda x}$	$\frac{\lambda^{\alpha-1}}{\Gamma(1-\alpha,\lambda x_{min})}$
exponential	$e^{-\lambda x}$	$\lambda e^{\lambda x_{min}}$

These results are quite surprising: we obtained a statistical distribution of contact opportunities that mimics what has been obtained for experiments in the *real world* [30, 66, 91]. It should be noted, however, that human activity roughly spans the 12 hours interval, while even the most assiduous user which we were able to track in our traces spent less than 4 consecutive hours on SL.

For the sake of brevity, there are a number of results that we omit from this Section: we studied the most important graph-theoretic properties of the radio networks emerging from contact opportunities among users in our traces, and we analyzed in more detail the characteristics of user movements in terms of distance covered during periods of mobility. The interested reader can refer to [69] for a comprehensive view of our measurement data. In summary, all our results indicate that the approach described in this Section represents a viable and elegant substitute to more complicated and costly experiments with real humans.

3.4 Summary

In the line of research presented in this Chapter, we made several contributions, that can be summarized as follows:

- We revisit traditional Location Theory and propose a distributed mechanism inspired by local search approximation algorithms (Section 3.1). Our solution exploits a formulation of a multi-commodity capacitated facility location problem to compute an approximate solution based on local measurements only. Through an extensive simulation study, we show that our scheme well approximates an optimal solution when both network and content dynamics are considered. Our mechanism achieves load balancing across the network and scales well with the network size, making it suitable for scenarios in which access congestion may appear. Furthermore, we compare our content replication scheme with existing mechanisms, and show under which conditions our approach yields better performance.

- We drop the (typical) assumption of a cooperative environment and formalize the problem of *selfish replication* (Section 3.2). Essentially, we assume users that operate their mobile devices to be rational and selfish when downloading content. As such, we model a game in which players are sensitive to content access delay, which is a function of the congestion level in the network and study both social optimum and equilibrium solutions for a simple instance (two players) of the game. The main findings of this approach is that an equilibrium (yet not efficient) strategy for selfish players is to randomize, in a way that bears substantial similarity to our distributed replication algorithm presented for a cooperative setting. Also, several open problems stem from this work, which require more elaborate equilibrium concepts (in game theoretic terms).

- We introduce a novel methodology to study human mobility and collect data that can be used in trace-driven experiments for the performance evaluation of a variety of mechanisms (routing, content distribution, etc...) that operate on mobile networks. Our contribution consists in a software component – called a *crawler* – that can be used to mine user activity on a popular virtual world, namely SecondLife. Using our crawler, we launch several measurement campaigns that allow us to study contact opportunities, network properties and mobility features of the avatars operated by SecondLife users. Our findings indicate that avatar mobility and traces thereof represent an efficient and realistic alternative to complex experiments to study human mobility in the real-world.

Chapter 4

On-line Content Storage

Contents

Introduction

This Chapter covers a selection of our work on the broad topic of on-line data storage. Similarly to what happened to content distribution for the Internet, the literature on content storage has followed a cyclic path: it started with a naive, client-server approach, attempted at solving outstanding issues due to a peer-to-peer (P2P) architecture, resorted to hybrid architectures (called, peer-assisted), and finally went back to a client-server approach, with the server-side being represented by "the Cloud".

On-line data storage has gained traction in the research community since its first incarnation, which deals with the design of a peer-to-peer (P2P) application for user data storage. Simply put, the idea of a P2P storage system is to use edge resources – such as storage space and bandwidth – available at the fringe of the Internet (that is end-host machines) to store user data. Due to the intermittent on-line behavior that characterize peers and because failures are the norm rather than the exception at such system scales, data availability and durability require special care. Coding techniques, that inject redundancy in the system, have been at the core of a large body of research in this domain, with a special focus on advanced techniques to mitigate bandwidth utilization.

In this context, the line of work discussed in this Chapter departs substantially from related work, in several aspects:

- We focus on a particular class of storage applications: data backup. While bearing some similarities with the storage counterpart, backup applications have salient characteristic (that we outline later) that make the design of P2P, hybrid and centralized architectures simpler: data access patterns are substantially different from typical storage workloads. We exploit this in several works discussed in this Chapter.

- We show that there are a series of important aspects that drive the behavior of on-line data storage applications that have been largely overlooked in previous work: namely, we focus on **scheduling problems**, and on **redundancy management**. For the former, the common approach in the literature is that of using random strategies, which we show to be inferior to scheduling decisions based on a-posteriori knowledge of peer behavior. For the latter, we show that – in the context of a backup application – it is possible to go beyond the largely adopted technique of computing redundancy in an off-line manner, and come up with an on-line approach that lowers resource strain on peers.

- We present a new an hybrid architecture, in which a "Cloud" storage service (e.g., servers in a data-center) is used as a temporary storage location (similarly to a forward buffer) to store user data, with the goal of improving data transfers performance with respect to a purely P2P architecture. Ultimately, data is deleted from the storage service when it is safely stored on remote peers.

For the sake of brevity, this Chapter omits several other results we obtained when studying the problem of non-cooperative data placement. This line of work, presented in [100–102], aims at finding equilibrium strategies when self-interested peers modulate the quantity (storage space) and quality (peer availability and bandwidth) of resources dedicated to the system with the aim of maximizing benefits (*i.e.,* the ability to store their data) and minimizing the costs (*i.e.,* the amount of resources they offer).

This Chapter also omits a measurement study we did on Wuala [78], a real-life and widely adopted on-line storage and backup application. In short, the objective of this recent work is to understand the architectural choices made by a real system, and explain its evolution from a peer-assisted application to a client-server service.

Before proceeding any further, we now overview the necessary background information and the application scenario that we use throughout the Chapter.

Background and Application scenario. Similarly to many on-line backup applications, we assume users (referred to as *data owners*) to specify one local folder containing important data to backup. Note that backup data remains available *locally* to data owners. This is an important trait that distinguishes backup from many on-line storage applications, in which data is only stored remotely.

We consider the problem of long-term storage of large, immutable, and opaque pieces of data that we term *backup objects*. Backup objects may be stored on remote peers, which are inherently unreliable. Peers may join and leave the system at any time, as part of their short-term on-line behavior: in the literature, this is referred to as *churn*. Moreover, peers may crash and possibly abandon the P2P application: this behavior is generally referred to as peer *death*. As such, the on-line behavior of peers must be continuously tracked, since it cannot be determined *a priori* [18].

The problem of achieving **data availability** when using peers with an intermittent on-line behavior has received ample attention in the past [17]. In our work, peers store their data in (encrypted) backup objects of size o. Each object is encoded in n *fragments* of a fixed size f which are ready to be placed on remote peers, or eventually, a storage server. Any k out of n fragments are sufficient to recover the original data backup. When using optimal erasure coding techniques, $k = \lceil o/f \rceil$. The redundancy management mechanism determines the redundancy level (or rate) $r = nf/o$. In this Chapter, we allow peers to upload s encoded blocks to a storage server (e.g., in a data-center with high availability), in addition to sending them to remote peers. Thus, $n = s + p(s)$.

The number of encoded blocks to upload to remote peers can be derived as follows. We consider the probability of each peer being on-line as an independent event with probability $\overline{a}$ (termed *peer availability*), and we aim for a data availability target value t. Therefore, we store on remote peers the number of fragments $p(s)$ defined as:

$$p(s) = \min \left\{ x \in \mathbb{N} \,\middle|\, \left(\sum_{i=k-s}^{x} \binom{x}{i} \overline{a}^{i} (1-\overline{a})^{x-i} \right) \geq t \right\} \tag{4.1}$$

The *redundancy rate* $r = \frac{(s+p(s))f}{o}$ represents the ratio between the quantity of data stored in the system (remote peers and storage server) and the original size of unencoded data. Now, suppose that a peer decides to store backup fragments on a remote server only: in this case, $p(s) = 0$ and $s = k$, thus the redundancy factor $r = 1$. Instead, assume a peer to store backup data on remote peers only: in this case $s = 0$ and p can be derived by the normal approximation to the binomial in Eq. 4.1 (see for example [17]). We note that, for Eq. 4.1 to hold, encoded fragments must be stored on *distinct remote peers*, because otherwise the failure of a single machine would imply the contemporary loss of many fragments, thereby resulting in a higher probability of data loss. On the other hand, since the data-center is considered to be highly available, any number of fragments can be stored on the data center.

Similarly to data availability, **data durability** can be achieved by injecting a sufficient level of redundancy in the system. One key issue to address is to determine the redundancy level required to make sure data is not lost, despite peer churn. This problem is called *redundancy management*. A closely related problem is to deal with peer deaths, which cause the data redundancy level to drop.

For the sake of clarity, we now explain the operation of a baseline P2P backup application. We gloss over the details of how data redundancy is achieved and discuss the salient phases of the life-time of backup data.

During the **backup phase**, data owners upload fragments to some selected remote peers. We assume that any peer can collect a list of remote peers with available storage space: this can be achieved with known techniques, e.g. a central coordinator or a decentralized data structure such as a distributed hash table. The backup phase completes when *all* n fragments are placed on remote peers.

Once the backup phase is completed, the **maintenance phase** begins. The purpose of this phase is to reestablish the desired redundancy level in the system, that may decrease due to peer deaths: new fragments must be re-injected in the system. The crux of data maintenance is to determine when the redundancy of the backup object is too low to allow data recovery and to generate other fragments to re-balance it. In the event of a peer death, the system may trigger the maintenance phase immediately (eager repairs) or may wait for a number of fragments to be tagged as lost before proceeding with the repairs (lazy repairs) [18,45,47]. As such, it is important to discern *unambiguously* permanent deaths from the normal on-line behavior of peers: this is generally achieved by setting a time-out value, Θ, for long-term peer unavailability.

Note that, as peers hold a local copy of their data, maintenance can be executed solely by the data owner, or (as often done in storage systems) it can be delegated. In both cases, it is important to consider the time-frame in which data cannot be maintained. First, fragments may be lost before a host failure is detected using the time-out mechanism outlined above. This problem is exacerbated by the availability pattern of the entity (data owner or other peers) in charge of the maintenance operation: indeed, host failures cannot be detected during the off-line periods. Second, data loss can occur during the restore process.

In the unfortunate case of a disk or host crash, the **restore phase** takes place. Data owners contact the remote machines holding their fragments, download at least k of them, and reconstruct the original backup data.

4.1 Scheduling Data Transfers

One of the most appealing characteristics of Peer-to-Peer storage and backup applications is that user data can be stored at low costs, using excess free capacity in local hard drives and/or removable devices; these applications require moving large amounts of data between end-hosts. The current state of technology implies that amounts of data that can comfortably fit on today's disk drives require a long time to be transferred: for example, on an ADSL line having a standard 1Mbps upload speed, 10 GB of data need almost a day of continuous upload. In addition, the unreliability of peers require data to be sent with redundancy and the fact that many nodes only spend a few hours on-line can further extend the amount of time needed to complete data transfers. Time to transfer has a strong impact over data durability: as long as data is not uploaded with redundancy to nodes, it risks being lost in the case of a local disk crash: in realistic scenarios, the most likely cause for data loss can be the simple fact that nodes experience a disk crash before completing the data upload.

The length of data transfers and their impact on data durability motivates the study of strategies to shorten them. As such, in this Section, we overview a new model of data transfers which we use to derive optimality results.

When nodes are homogeneous, in terms of bandwidth and connectivity behavior, scheduling choices are not significant. Conversely, we show that the peer heterogeneity observed in real applications makes scheduling matter, since informed data transfer policies can avoid a situation where uploads would stall. In related work, peer connectivity patterns are usually taken into account using simple mathematical modeling, often using memory-less processes where the probability that a node disconnects is the same for each node and at each time. In this case, the scheduling problem becomes trivial: since all nodes have the same behavior, choosing one or the other is indifferent. However, since it is guided by human behavior, churn is not a completely random process: node availability exhibits regularities such as diurnal and weekly patterns, and different behavior between users [71]. Trying to create more complex churn models that attempt to describe all the particularities and regularities of user behavior would be prohibitive, given the inherent complexities of human behavior. We instead take a different approach, using availability traces *as input* of the scheduling problem.

Next, we formally define the scheduling problem and study optimal scheduling based on *a priori* knowledge of peer behavior: this provides a baseline performance figure against which we compare feasible on-line scheduling choices, which instead are based on *a posteriori* knowledge. While the problem may appear difficult to solve in polynomial time, we show that the optimal scheduling can be calculated efficiently.

In our work, we also study the problem of scheduling with an experimental approach. We thus build a discrete, event-based simulator that features, in addition to the components in charge of data transfers, an implementation of the *peer selection* process that is used to select a list of remote peers – which we label *peer set* – eligible for storing data. Based on trace-driven simulation results (that we omit for the sake of space), we conclude that the most important factor influencing the time needed to complete uploads is the number of nodes present in peer sets: as this value grows, the time to complete transfers decreases rapidly. This is an important message to application designers: allowing a small degree of flexibility for the choice of nodes to adopt in the overlay pays off significantly. In addition, we discover that congestion has only a moderate impact on the amount of time needed to complete data transfers, imposing small penalties with respect to cases where a single node is sending data, mainly due to asymmetric up- and down-links. Finally, we show that our on-line scheduling heuristics help significantly in reducing data transfer times, with a decrease in the overhead due to non-optimal scheduling by a factor of around 40% in our experiments.

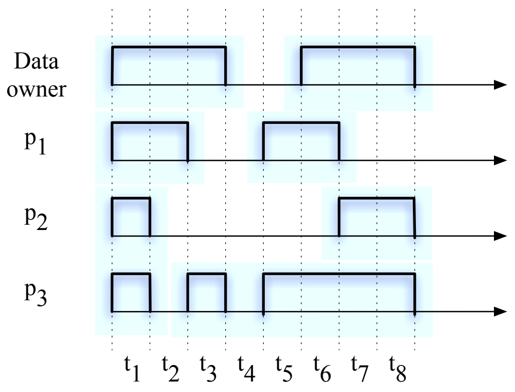


Figure 4.1: Example availability traces.

Symbol	Meaning in the upload scenario	Meaning in the erasure-coded download scenario
n	peer set size	number of remote data holders
T	number of time-slots	number of time-slots
o	data object size	data object size
$d_{i,t}$	data peer i can download in time-slot t (0 if i is off-line)	data peer i can upload in time-slot t (0 if off-line)
u_t	data owner can upload in time-slot t (0 if owner is off-line)	data owner can download in time-slot t (0 if off-line)
m_i	maximum amount of data uploaded to peer i	data stored on peer i

Table 4.1: A summary of notation used in the scheduling problem.

4.1.1 The Problem

The way scheduling is chosen can obviously impact the amount of time needed to complete a data transfer. Consider the availability traces in Figure 4.1, where the data owner has a unitary upload speed per time-slot, and has to upload one data unit per remote peer – one each to p_1, p_2 and p_3. With an optimal schedule the owner would send a unit to p_2 in the first time-slot, then one to p_1 in time-slot t_2, then one to p_3 concluding the transfer in time-slot t_3. Conversely, if data is sent to p_3 during the first time-slot, in the second time-slot p_1 is the only possible choice. The transfer will have to stall until p_2 comes back on-line in time-slot t_7.

More generally, a data owner has a peer set of n nodes to which it needs to upload a data object of size o. We take traces of peer up-times as an input of our problem, encompassing T time-slots (starting from the beginning of the upload process) in which peer availability and bandwidth can change; within a single time-slot the network conditions remain stable. In our model, the time-slot duration is not constrained to be constant.

We model *network* bottlenecks as imposed by the upload/download bandwidth of peer access links, and we assume that bandwidth can vary between time-slots: we thus model u_t as the amount of data that the owner can upload within time-slot t, and $d_{i,t}$ as the amount of data that peer i can download in the same time-slot t. We express the fact that nodes are off-line in a time-slot by setting the corresponding bandwidth to 0.

We impose an additional constraint m_i on the maximum data that can be uploaded to each peer i. This restriction can be due to both storage capabilities of nodes and to system design choices (for example, if the data owner is uploading the result of an erasure coding process, having too much data on the same node could degrade data availability or durability).

Next, we examine the problem of finding an optimal schedule that minimizes the time to transfer of a *single* node. We refer the reader to [99] for a comprehensive study of the scheduling problem in which we also present an extension to the basic model presented here, which addresses the problem of "storage congestion" that arises when multiple peers upload data concurrently. Furthermore, [99] also discusses how to reformulate the problem for download operations.

The notation we use in the following is summarized in Table 4.1.

Definition 2 *The* ideal time to transfer *($\tilde{\xi}$) is the minimum number of time-slots needed to upload the data object considering only the bandwidth of the data owner:*

$$\tilde{\xi} = \min \left\{ t \in 1 \ldots T : \sum_{i=1}^{t} u_i \geq o \right\}.$$

$\tilde{\xi}$ represents the time to transfer data when uploading to an ideal server, which is supposed to be always on-line and to have enough bandwidth to saturate the owner's up-link. The differences between $\tilde{\xi}$ and time to transfer values observed for P2P systems are entirely due to the limits of remote nodes and to the inefficiency of scheduling policies.

Definition 3 *A* schedule *S represents the amount of data sent to each node during each time-slot. For each peer i and time-slot t, we will denote $S(i,t)$ as the amount of data sent to peer i during time-slot t. During a time-slot t, a node can send data concurrently to several peers at once, reflecting the case where a node uploads data in parallel to several destinations. A schedule has to satisfy the following conditions.*

- ***Upload constraints**: $\forall t \in [1,T] : \sum_{i=1}^{n} S(i,t) \leq u_t$.*
- ***Download constraints**: $\forall i \in [1,n], t \in [1,T] : S(i,t) \leq d_{i,t}$*
- ***Storage constraints**:$\forall i \in [1,n] : \sum_{t=1}^{T} S(i,t) \leq m_i$*

We denote the set of all schedules as $\mathcal{S}$.

Definition 4 *A schedule S is* complete *if at least a total amount o of data has been transmitted:*

$$\sum_{i=1}^{n}\sum_{t=1}^{T} S(i,t) \geq o. \tag{4.2}$$

We denote the set of all complete schedules as $\mathcal{CS}$.

Definition 5 *The* time to transfer *(ξ) of a schedule S is its completion time, i.e. the last time-slot in which the data owner uploads data:*

$$\xi(S) = \max\left\{ t \in [1,T] : \sum_{i=1}^{n} S(i,t) > 0 \right\}.$$

The goal of a scheduling policy is to obtain the shortest possible ξ.

4.1.1.1 Optimal Scheduling

We now assume peer availability traces to be known, and show how to compute the minimum time it takes to upload data to remote peers. Despite the fact that finding optimal scheduling may appear computationally very expensive at first sight, we devise an efficient polynomial-time solution based on a max-flow formulation.

Definition 6 *The* optimal time to transfer *($\mathring{\xi}$) is the minimum ξ within the set of all complete schedules $\mathcal{CS}$:*

$$\mathring{\xi} = \min\left\{\xi(S) : S \in \mathcal{CS}\right\}. \tag{4.3}$$

In the following, we use $\mathring{\xi}$ as a baseline to compute the overhead in time-to-transfer for a given scheduling policy.

Definition 7 *The* scheduling overhead *for a schedule S is the relative increase in ξ due to a non-optimal scheduling:*

$$\frac{\xi - \mathring{\xi}}{\mathring{\xi}}.$$

Now, we compute optimal scheduling by solving several instances of the related problem: "how much data can be transferred within the first $\bar{t}$ time-slots"? We will use the following Proposition to relate the two problems.

Proposition 1 *Let $\mathcal{S}$ be the set of all schedules, and $F(\bar{t})$ be the maximum amount of data that can be uploaded not later than $\bar{t}$, that is:*

$$F(\bar{t}) = \max\left\{ \sum_{i=1}^{n}\sum_{t=1}^{\bar{t}} S(i,t) : S \in \mathcal{S} \wedge \xi(S) \leq \bar{t} \right\}; \tag{4.4}$$

$\mathring{\xi}$ will be:

$$\mathring{\xi} = \min\left\{\bar{t} \in 1 \ldots T : F(\bar{t}) \geq o\right\}. \tag{4.5}$$

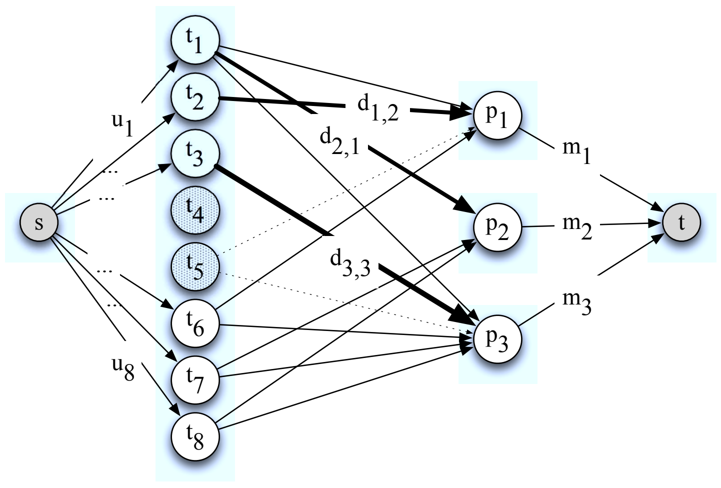


Figure 4.2: Flow network equivalent to the traces of Figure 4.1.

Proof: Let $t_1 = \mathring{\xi}$ and $t_2 = \min\left\{\bar{t} \in 1 \ldots T : F(\bar{t}) \geq o\right\}$. We show that both $t_1 \geq t_2$ and $t_1 \leq t_2$ hold.

1. $t_1 \geq t_2$. By Equation 4.3, an $S_1 \in \mathcal{CS}$ exists such that $\xi(S_1) = t_1$ and, since $S_1 \in \mathcal{CS}$, by Equation 4.2 $\sum_{i=1}^{n} \sum_{t=1}^{T} S(i,t) \geq k$. The existence of S_1 implies that $F(t_1) \geq o$ (Equation 4.4) and therefore $t_2 \leq t_1$.

2. $t_1 \leq t_2$. By Equation 4.4, an S_2 exists such that $\xi(S_2) = t_2$ and $\sum_{i=1}^{n} \sum_{t=1}^{T} S(i,t) \geq o$. This directly implies that $t_1 = \mathring{\xi} \leq t_2$.

■

The former Proposition allows us to find $\mathring{\xi}$ by computing different values of $F(t)$ and by finding the smallest value $\bar{t}$ such that $F(\bar{t}) \geq o$.

4.1.1.2 Max-flow Formulation

Let us now focus on how to compute $F(t)$. This problem can now be encoded as a max-flow problem on a network built as follows. First, we create a complete bipartite directed graph $G' = (V', E')$ where $V' = \mathcal{T} \cup \mathcal{P}$ and $E' = \mathcal{T} \times \mathcal{P}$; the elements of $\mathcal{T} = \{t_i : i \in 1 \ldots T\}$ represent time-slots, the elements of $\mathcal{P} = \{p_i : i \in 1 \ldots n\}$ represent remote peers. Source s and sink t nodes are then added to the graph G' to create a flow network $G = (V, E)$. The source is connected to all the time-slots during which the data owner is on-line; all peers are connected to the sink.

The capacities on the edges are defined as follows: each edge from the source s to time-slot i has capacity u_i; each edge between time-slot t and peer i has capacity $d_{i,t}$; finally, each edge between peer i and the sink has capacity m_i.

Since we are interested in maximal flow, we can safely ignore (and remove from the graph) those edges with capacity 0 (corresponding to nodes that are off-line). In Figure 4.2, we show the result of encoding the example of Figure 4.1.

We show that each $s \to t$ network flow represent a schedule, and a maximal flow represents a schedule transferring the maximal data $F(t)$. In the example of Figure 4.2, the bold edges represent a solution to the maximal flow problem on the first 3 time-slot nodes where an amount of data $o = 3$ is uploaded, with a flow of 1 per edge.

A nonzero flow from a time-slot node to a peer node represents the data uploaded towards that node in the specific time-slot; parallel transfers happen when multiple outgoing edges from a single time-slot node have nonzero flow. The constraints that guarantee that the schedule is valid according to Definition 3 are guaranteed by the edge labels: upload constraints are guaranteed by edges from source to time-slot; download constraints by edges from time-slots to peers ; storage constraints by edges from peers to the sink.

4.1.1.3 Computational Complexity

Algorithm 1 Algorithm for finding $\mathring{\xi}$.

```
$l \leftarrow 1$; $r \leftarrow 1$
% We look for a $r$ value with $D(r) \geq o$.
% In this cycle, maximum $\log_2 \bar{t}$ invocations to $D$.
while $D(r) < o$:
    $l \leftarrow r$; $r \leftarrow 2r$
% Now $l \leq \bar{t} \leq r$; we look for $\bar{t}$ via binary search.
% Again, maximum $\log_2 \bar{t}$ invocations to $D$.
while $l \neq r$:
    $t \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$
    if $D(t) < o$:
        $l \leftarrow t$
    else:
        $r \leftarrow t$
return $l$
```

As guaranteed by Proposition1, optTTT can be obtained by finding the minimum value $\bar{t}$ such that $F(\bar{t}) \geq o$. The $\bar{t}$ value can be found by binary search, requiring $O\left(\log \bar{t}\right)$ calls to the routine computing F as in Algorithm 1; for a flow network with V nodes and E edges, the max-flow can be computed with time complexity $O\left(VE \log\left(\frac{V^2}{E}\right)\right)$ [56]. In our case, when we have n nodes and an optimal solution of $\bar{t}$ time-slots, V is $O(n+\bar{t})$ and E is $O(n\bar{t})$. The complexity of an instance of the max-flow algorithm is thus $O\left(n\bar{t}\left(n \log \frac{n}{\bar{t}} + \bar{t} \log \frac{\bar{t}}{n}\right)\right)$. Multiplying this by the $O\left(\log \bar{t}\right)$ times that the max-flow algorithm will need to be called, we obtain a computational complexity for the whole process of $O\left(n\bar{t} \log n \left(n \log \frac{n}{\bar{t}} + \bar{t} \log \frac{\bar{t}}{n}\right)\right)$.

4.1.2 On-line Scheduling Policies

As opposed to the optimal scheduling considered until now, we now move on to discuss strategies that can can actually be implemented, meaning that a scheduling decision applied at time t is only dependent on information that is available at time t. For convenience, we use $a_{i,t}$ as a binary value assuming value 1 if peer i is on-line at time-slot t: $a_{i,t} = 1$ if $d_{i,t} \neq 0$, 0 otherwise.

Each of the scheduling policies we introduce in this Section gives a priority value $v_i(t)$ to each node i at time t. The scheduling policy chooses to upload data to the available node in the peer set with the highest priority value. In case of ties, we break them by selecting nodes randomly. If the highest-priority node is unavailable or the upload speed of the data owner is not saturated, further nodes are selected by descending order of priority.

- **Random Scheduling:** The simplest scheduling choice, which is most commonly used in existing systems, amounts to just choosing a node at random within the peer set: $v_i(t) = 0$. Since all nodes will be tied in term of priority, scheduling will be chosen randomly. Random scheduling is extremely cheap and easy to implement because it is *stateless*: no information has to be kept about past node behavior.

- **Least Available First:** A data transfer can stall if nodes that should receive the next pieces of data are not available. This strategy is based on assuming that nodes that have been on-line often in the past will continue to do so in the future; it thus makes sense to prioritize uploads towards nodes that have been less available in the past: when only high-availability nodes are on-line, data stored on them will be less likely to have already reached the maximum value m_i. This scheduling policy observes past availability within a "window of past behavior" lasting for w time-slots: $v_i(t) = -\sum_{x=t-w}^{t} a_{i,x}$.

- **Slowest First:** This is a variant of the least-available-first policy, also taking into account the download speed of nodes, based on the idea that a node with slower download speed will complete receiving its maximal amount of data m_i in longer time: $v_i(t) = -\sum_{x=t-w}^{t} d_{i,x}$.

- **Last Connected First:** If the amount of time that nodes spend on-line is exponentially distributed, each node has the same probability of going off-line independently of the amount of time spent on-line until the present. On the contrary, different distributions are observed in practice. In particular, if nodes that have been on-line for longer are more likely than others to remain on-line, it makes sense to prioritize uploads towards nodes that connected most recently, in order to capitalize on the capability of uploading to them before they disconnect: $v_i(t) = \max\{x \in [1, t] : a_{i,t} = 0\}$.

- **Longest Connected First:** If, as opposed to what has been discussed before, the amount of time a node spends on-line tends to be more concentrated towards the mean than in an exponential distribution, it makes sense to prioritize uploads towards node that got connected least recently: $v_i(t) = -\max\{x \in [1,t] : a_{i,t} = 0\}$.

4.1.3 Results

In this work, we study the performance of various scheduling policies using real application traces. Besides using realistic up-link/down-link capacities associated to the peers of our experiments, we extract availability traces (*i.e.*, logon/logoff events) from the logs of an instant messaging (IM) server that last for a duration of 3 months. The reason why we believe such traces to be representative of the on-line behavior of peers is that in both IM and on-line storage, users are generally signed in for as long as their machine is connected to the Internet.

Via simulation, we obtained various insights: for a detailed overview of the complete simulation setup, definition of performance metrics and experimental results for a variety of parameters, we refer the reader to [99]. Here we discuss our main findings:

- As the number of nodes in peer sets grows, the time to complete transfers decreases rapidly. This is an important message to application designers: allowing a small degree of flexibility with respect to the choice of nodes to adopt in the overlay for storing data pays off significantly.
- A simple scheduling policy such as Least Available First manages to cut around 40% of the scheduling overhead, based on the assumption that nodes that have been on-line often in the past will continue to do so in the future.
- The overhead with respect to optimal scheduling is very unevenly distributed: many nodes will barely experience a difference between the optimal schedule and the one they took in practice, but for a relevant percentage of them there is the possibility that bad schedule choices will result in much longer data transfer times.
- When many nodes are uploading data at the same moment, congestion only has a small impact on transfer completion times.
- The impact of scheduling in the case of downloads is much less significant, due to the asymmetry of peers' bandwidth.

The performance obtained by a scheduling algorithm is a consequence of the predictability of the connectivity patterns of users. If their connections and disconnections could be forecast with certainty, there would be a way to devise optimal scheduling; our simple scheduling policies can be thought of as "guessing" future node connectivity, and they manage to reduce the time needed to complete transfers.

An important open avenue for research, that we are currently investigating, is the study of sophisticated techniques to predict user availability, and apply such knowledge to the scheduling problem.

4.2 Redundancy Management

As hinted in the introduction to this Chapter, for backup applications, the focus shifts from data availability to durability, which amounts to guaranteeing that data is not lost. Furthermore, the requirements for a specialized backup application are less stringent than those of generic storage in several aspects:

- data backup often involves the bulk transfer of potentially large quantities of data, both during regular backups and, in the event of data loss, during restore operations. Therefore, read and write latencies of hours have to be tolerated by users;
- data owners have access to the original copy of their data (which is stored in their local hard drives), making it easy to inject additional redundancy in case data stored remotely is partially lost;
- since data is read only during restore operations, the application does not need to guarantee that any piece of the original data should be promptly accessible in any moment, as long as the time needed to restore the whole backup remains under control.

In this Section, we overview the design of a new *redundancy management* mechanism tailored to backup applications. Simply stated, the problem of redundancy management amounts to computing the necessary redundancy level to be applied to backup data to achieve durability. The endeavor of this work is to design a mechanism that achieves data durability without requiring high redundancy levels nor fast mechanisms to detect node failures. Our solution to the problem stems from the particular data access workload of backup applications: data is written once and read rarely. The gist of our redundancy management mechanism is that the redundancy level applied to backup data is computed in an *on-line* manner. Given a time window, that accounts for failure detection and data repair delays, and a system-wide statistic on peer deaths, a peer determines the redundancy rate during the backup phase. A byproduct of our approach is that, if the system state changes, then peers can adapt to such dynamics and modify the redundancy level on the fly.

The ability to compute the redundancy level in an *on-line* manner requires solving several problems related to coding efficiency and data management: in this Section we show how our scheme can be realized in practice, exploiting the properties of Fountain Coding.

Finally – as we did for other works described in this Chapter – we evaluate our redundancy management scheme using trace-driven simulations. In this Section we discuss our main results, which indicate that our approach drastically decreases strain on peer resources, reducing the storage and bandwidth requirements by a factor between two and three, as compared to redundancy schemes that use a fixed, system-wide redundancy factor. This result yields augmented storage capacity for the system and decreased backup times, at the expense of increased restore times, which is a reasonable price to pay if the specific requirements of backup applications are taken into account.

4.2.1 The problem

Before proceeding with a formal problem statement, we need to clarify the performance metrics we are interested in for this work. Overall, we compute the performance of a P2P backup application in terms of the amount of time required to complete the backup and the restore phases, labeled *time to backup* (TTB) and *time to restore* (TTR).[1] Moreover, in the following, we use baseline values for backup and restore operations which bound both TTB and TTR. We compute such bounds as follows: let us assume an *ideal* storage system with unlimited capacity and uninterrupted on-line time that backs up user data. In this case, TTB and TTR only depend on the size of a backup object and on up-link bandwidth and availability of the data owner. We label these ideal values $minTTB$ and $minTTR$.[2] Formally, we have that a peer i with upload and download bandwidth u_i and d_i, starting the backup of an object of size o at time t, completes its backup at time t', after having spent $\frac{o}{u_i}$ time on-line. Analogously, i restores a backup object with the same size at t'' after having spent $\frac{o}{d_i}$ time on-line. Hence, we have that $minTTB(i,t) = t' - t$ and $minTTR(i,t) = t'' - t$. We use these reference values throughout the paper to compare the relative performance of our P2P application versus that of such an ideal system.

To complement the problem definition, and with reference to the application scenario discussed in the Introduction to this Section, we consider a redundancy management scheme whose objective is to ensure data is not lost in a well-defined time-window $w = \Theta + a_{off}$, where a_{off} is the (largest) transient off-line period of the entity in charge of data maintenance. For example, if the data owner executes data maintenance: first, it needs to be on-line to generate new fragments and upload them, and second, the timeout Θ has to be expired.

Our objective is to design a redundancy management mechanism to achieve data durability: in practice, data can be considered as durable if the probability to lose it, due to the permanent failure of hosts in the system, is negligible. Hence, the problem of designing a system that guarantees data durability can be approached under different angles.

As noted in previous works [46, 86], data availability implies data durability: a system that injects sufficient redundancy for data to be available at any time, coupled with maintenance mechanisms, automatically achieves data durability. These solutions are, however, too expensive in our scenario: the amount of redundancy needed to guarantee availability is much higher than what needed to obtain durability.

[1] TTB and TTR are clearly related to the time to transfer ξ, discussed in Section 4.1. Here we change notation to explicitly take into account up-link and down-link data transfers.

[2] $minTTB$ and $minTTR$ are clearly related to the ideal time to transfer $\hat{\xi}$ presented in Section 4.1. Here we change, again, to account for ideal up-link and down-link data transfers.

Instead of using high redundancy, data durability can also be achieved with efficient maintenance techniques. For example, in a data-center, each host is continuously monitored: based on statistics such as the mean time to failure of machines and their components, it is possible to store data with very little redundancy and rely on system monitoring to detect and react immediately to host failures. Failed machines are replaced and data is rapidly repaired due to the dedicated and over-dimensioned nature of data-center networks. Unfortunately, this approach is not feasible in a P2P setting. First, the interplay of transient and permanent failures makes failure detection a difficult task. Since it is difficult to discern deaths from the ordinary on-line behavior of peers, the detection of permanent failures requires a delay during which data may be lost. Furthermore, data maintenance is not immediate: in a P2P application deployed on the Internet, bandwidth scarceness and peer churn make the repair operation slow.

In summary: on the one hand durability could be achieved with high data redundancy, but the cost in terms of resources required by peers would be overwhelming. On the other hand, with little redundancy, durability could be achieved with timely detection of host failures and fast repairs, which are not realistic in a P2P setting.

The endeavor of this work is to design a redundancy management mechanism that achieves data durability without requiring high redundancy levels nor fast failure detection and repair mechanisms. Our solution to the problem stems from the particular data access workload of backup applications: data is written once, during backup, and read (hopefully) rarely, during restores. Hence, we design a mechanism that injects only the data redundancy level required to compensate failure detection and data repair delays. That is, we define data durability as follows.

Definition 8 *Data durability d is the probability to be able to access data after a time window t, during which no maintenance operations can be executed.*

Definition 9 *The time window t is defined as $t = w + TTR$, where w accounts for failure detection delays and TTR is the time required to download a number of fragments sufficient to recover the original data.*

Let's recall that w depends on whether the maintenance is executed by the data owner or is delegated, and can be thought of a parameter of our scheme.

The goal of our redundancy management mechanism is to determine the data redundancy that achieves a target data durability: we proceed as follows. A peer with n fragments placed on remote peers could lose its data if more than $n - k$ of them would get lost as well within the time window t. The data redundancy required to avoid this event, is $r = n/k$. Now, let us assume peer deaths to be memoryless events, with constant probability for any peer and at any time. Peer lifetimes are exponentially distributed stochastic variables with a parametric average τ. Hence, the probability for a peer to be alive after a time t is $e^{-t/\tau}$. Assuming death events are independent, data durability writes as:

$$d = \sum_{i=k}^{n} \binom{n}{i} \left(e^{-t/\tau}\right)^{i} \left(1 - e^{-t/\tau}\right)^{n-i}. \tag{4.6}$$

Eq. 4.6 depends on t which, in turn, is a function of TTR. However, peers cannot readily compute their TTR, as this quantity depends on the characteristics of remote peers hosting their fragments. We thus propose to use the following heuristic as a method to *estimate* the TTR. Suppose peer p_0 is computing an estimate of its TTR. In the event of a crash, we assume p_0 to remain on-line during the whole restore process. In such a case, assuming no network bottlenecks, its TTR can be bounded for two reasons: *i)* the download bandwidth D_0 of peer p_0 is the bottleneck; *ii)* the upload rate of remote peers holding p_0's data is the bottleneck. Let us focus on the second case: we define the *expected upload rate* μ_i of a generic remote peer p_i holding a backup fragment of p_0 as the product of the availability a_i of peer p_i and its upload bandwidth, that is $\mu_i = u_i a_i$.

Peer p_0 needs to download at least k fragments to fully recover a backup object. Let us assume these k fragments are served by the k remote peers with the highest expected upload rate μ_i. In this case, the "bottleneck" is the k-th peer with the lowest expected upload rate μ_k. Then, an estimation of TTR, that we label $eTTR$, can be obtained as follows:

$$eTTR = \max\left(\frac{o}{D_0}, \frac{o}{k\mu_k}\right). \tag{4.7}$$

We now set off to describe how our redundancy management scheme works in practice: the redundancy level applied to backup data is computed by the combination of Eq. 4.6 and Eq. 4.7. Let us assume, for the sake of simplicity, the presence of a central coordinator that performs membership management of the P2P network: the coordinator keeps track of users subscribed to the application, along with short-term measurements of their availability, their (application-level) up-link capacity and the average death rate τ in the system. While a decentralized approach to membership management and system monitoring is an appealing research subject, it is common practice (*e.g.*, Wuala[3]) to rely on a centralized infrastructure and a simple heartbeat mechanism.

During a backup operation, peers query the coordinator to obtain remote hosts that can be used to store fragments, along with their availability. A peer constructs a backup object, and subsequently uploads k fragments to distinct, randomly selected available remote hosts. Then the peer continues to inject redundancy in the system, by sending additional fragments to randomly selected available peers, until a stop condition is met. Every time one (or more) new fragment is uploaded, the peer computes d and $eTTR$: the stop condition is met if $d \geq \sigma_1$ and $eTTR \leq \sigma_2$. While selecting an appropriate σ_1 is trivial, in the following we define σ_2 as $\sigma_2 = \alpha \cdot minTTR$, where α is a parameter that specifies the degradation of TTR with respect to an ideal system, tolerated by users.

[3]http://www.wuala.com

We now discuss in details the influence of the two stop conditions on the behavior of our mechanism. Given Eq. 4.6, we study the impact of the ratio $\frac{w+eTTR}{\tau}$:

- $\tau \gg w + eTTR$: this case is representative of a "mature" P2P application in which the dominant factor that characterizes peer deaths are permanent host failures, rather than users abandoning the system. Hence, the exponential in Eq. 4.6 is close to 1, which implies that the target durability σ_1 can be achieved with a small n.

  As such, the condition on $eTTR \leq \sigma_2$ prevails on $d \geq \sigma_1$ in determining the redundancy level to apply to backup data. This means that the accuracy of the estimate $eTTR$ plays an important role in guaranteeing acceptable restore times; instead, errors on $eTTR$ have no impact on data durability.

- $\tau \sim w + eTTR$: this case is representative of a P2P application in the early stages of its deployment, where the abandon rate of users is crucial in determining the death rate. In this case, the exponential in Eq. 4.6 can be arbitrarily small, which implies that $n \gg k$, *i.e.,* the target durability d requires higher data redundancy.

  In this case, the condition $d \geq \sigma_1$ prevails on $eTTR \leq \sigma_2$. Hence, estimation errors on the restore times may have an impact on data durability: *e.g.*, underestimating the TTR may cause n to be too small to guarantee the target σ_1.

In summary, the key idea of our redundancy management mechanism is that the redundancy level applied to backup data is computed in an *on-line* manner, during the backup phase. This comes in sharp contrast to computing the redundancy level in an *off-line* manner, solely based on system-wide statistics, that characterize previous approaches to redundancy management.

A by-product of our approach is that our mechanism can *adapt* the redundancy rate r each peer applies to its data based on system dynamics. Now, we must prove that the system reaches a *stable state*: system dynamics must not bring the redundancy mechanism to oscillate around r. Based on Eq. 4.6 and Eq. 4.7, we face a retroactive system in which a feedback loop exists on the durability d. Given a target durability d, a system-wide average death rate τ and a time window $t = w + eTTR$, we can derive r. The problem is that eTTR depends on the short-term behavior of peers as well as the redundancy rate r.

First, we study how eTTR and d vary as a function of the redundancy rate r.

Proposition 2 *$eTTR$ is a non-increasing function in r.*

Sketch of the proof: Recall that $r = \frac{nf}{o}$. Let us assume a peer p_0 has the following ranked list of remote peers: $\{\mu_1, \mu_2, \mu_3, ..., \mu_k\}$, where, without loss of generality, $\mu_i < \mu_j \, \forall i < j$. If r increases, then n increases: new fragments must be stored on new remote peers. For simplicity, assume a single fragment is to be placed on peer p_q. Two cases can happen: *(i)* $\mu_q > \mu_k$; in this case, eTTR remains unvaried, since

p_q is "slower" than the k-th peer used to compute $eTTR$; *(ii)* $\mu_q < \mu_k$; in this case, p_q "ejects" the current k-th peer from the ranked list defined above. As such, $eTTR$ can only decrease. Note that eTTR may not reach the stop condition σ_2 if the parameter α is not appropriately chosen: simply stated, a *plateau* value of eTTR exists when placing fragments on all peers in the network.

Proposition 3 *d is an increasing function in r.*

Sketch of the proof: Eq. 4.6 is a composite function of eTTR. Hence, by increasing r, new fragments have to be placed on remote peers and it is not guaranteed, in general, that this contributes to decrease d. However, thanks to Proposition 2, eTTR is non-decreasing in r, hence $t = w + eTTR$ is non decreasing in r. As a consequence, d is an increasing function in r.

We can now state the following Proposition:

Proposition 4 *The redundancy management mechanism presented in this section is stable.*

Sketch of the proof: By design, our redundancy mechanism shall only increase r. Now, Proposition 2 states that increasing r yields lower values of $eTTR$, hence, eventually, the system either arrives at the stop condition $eTTR \leq \sigma_2$, when α is chosen appropriately, or it reaches the plateau defined above. Similarly, by Proposition 3, increasing the redundancy in the system implies that d grows asymptotically to 1, hence the system eventually reaches the stop condition $d \geq \sigma_1$.

It is natural to question why in Proposition 4 we omit the possibility of removing fragments from remote peers if r is too high. Let us consider such an operation: one possibility would be to drop a remote fragment at random. This operation would be unstable: indeed, for example, deleting a fragment from the "fastest" peer in the ranked list defined above would increase eTTR, decrease d, which as a consequence might require to re-inject a fragment. Instead, we could delete fragments starting from the "slowest" peer: in this case, the drop operation would be stable, but the storage load in the system may eventually become concentrated on fast peers only. Moreover, avoiding deletions can spare maintenance operations in the future should one or more of the remaining fragments on remote peers be lost. Due to these reasons, in this work we do not allow fragments to be dropped.

4.2.2 Coding and Data Management

With the redundancy management mechanism described above, the redundancy level applied to backup data is computed in an *on-line* manner. Instead, the redundancy rate used in most related work is usually computed *off-line*, given sufficiently representative statistics on the system, including transient and non-transient failures. These system-wide statistics are used to compute a unique redundancy rate that every peer will use. Instead, our approach requires each peer to compute an *individual* redundancy level: the time window t is a function of $eTTR$, which is different for every peer. In this Section, we study the practical implications that stem from adopting an on-line approach to redundancy management.

We argue that Fountain Codes [76] are a natural choice for our use case, because of their unique characteristics. With Fountain Codes, the generation of any fragment is independent from other redundant fragments (*on-the-fly* property) and the number of fragments that can be generated from the original data is potentially infinite (*rateless* property). Fountain Codes can be readily applied to the our mechanism: as long as the conditions on eTTR and d are not met, the encoder continues to generate new unique fragments *on the fly*. When the stop condition is reached, the encoding process terminates. Moreover, in case system dynamics trigger the generation of new encoded fragments (*e.g.*, because host availability decreases), these can be simply generated as needed, with the same procedure described above.

Coding techniques, including Fountain Coding, split the original data in *blocks* and encode each one separately. Blocks are seen as a sequence of *symbols* and encoded in a set of *codewords*. It is important to note, here, that the data transfer unit – in our terminology, a *fragment* – can contain one or more codewords. In particular, with Fountain Coding it is advisable to use rather small codewords, resulting in fragments containing a large number of them.

Note also that Fountain-encoded codewords are statistically "interchangeable": any codeword can be used to reconstruct the original data and any codeword can be replaced by any newly generated codeword. As a consequence, Fountain Codes also benefit the maintenance operation: indeed, peers need not track of the exact set of codewords to replace, which results in simplified book-keeping operations.

In this context, we note that an appealing characteristic of Fountain Codes is that the block size is *not constrained*: as opposed to alternative coding algorithms, the block size is not imposed by the "mathematical" construction of the coding algorithm itself, and can be defined to accommodate the characteristics of the backup data. Indeed, in addition to efficiency and complexity considerations, we note that the block size can be set as a function of the rate at which users generate backup data.

A shortcoming of Fountain Codes is that they are not optimal, in the sense that the amount of encoded data necessary to perform a restore is slightly larger than the size of the original backup object. There is a trade-off between this coding inefficiency and the computational requirements of the coding scheme, which depends on number of symbols per block. In recent years much effort has been spent in improving the performance of Fountain Codes to this respect. A possibility is to use a sliding-windowing approach [21, 28], which can increase coding efficiency whilst maintaining smaller block size. This approach "virtually" increases the encoding block by allowing the overlap of two or more subsequent coding blocks (referred as "windows"). The block overlap is a design parameter that impacts the performance of the code and its value can be decided *a priori* or according to customized coding strategies. For example, a classic LT code block of 40,000 symbols requires a redundancy factor of 1.07 to be correctly decoded with probability 0.9, whilst using a sliding-window approach (with overlap of 87.5%) the redundancy drops to 1.01 [21].

We remark that we have implemented a library in C (with Python wrappers) implementing LT codes [76] and including the sliding-window approach described above. This library is a core component of a prototype P2P backup application that we built.

4.2.3 Results

We now discuss the main findings of our experiments, that we obtained with a trace-driven system simulation. A comprehensive presentation of all our results is available in [99]. Our focus is to study the time required to backup and restore user data and perform a comparative analysis of the results achieved by a system using our redundancy management scheme and the traditional approach used for storage applications. For the latter case, we implement a technique in which the coding rate is set once and for all based on a system-wide average of host availability.

Note that, for the purpose of our study, it is not necessary to implement in detail a coding mechanisms: all we need to know for the evaluation of transfer times is the number of fragments each peer has to upload during the backup operation.

As we did for the Scheduling problem, we use traces as input to our simulator that cover both the on-line behavior of peers and their up-link and down-link capacities. Instead, long-term failures and the events of peers abandoning the applications, which constitute the peer deaths, follow a simple model driven by the parameter τ, as explained above. Due to the lack of traces that represent the realistic "data production rate" of Internet users, in this work we confine our attention to a homogeneous setting: each user has an individual backup object of the same size.

Overall, our experiments show that, in a realistic setting, a redundancy that caters to data durability can be less than half of what is needed to guarantee availability. This results in a system with a storage capacity that is more than doubled, and backup operations that are much faster (up to a factor of 4) than on a backup system based on traditional redundancy management techniques. This latter property is particularly desirable since, in most of the cases, peers suffering data loss were those that could not complete the backup before crashing.

Our results also pinpoint that the price to pay for efficient backup operations is a decreased (but controlled) performance of restore operations. We argue that this is a reasonable penalty, considering that all peers in the system would benefit from backup efficiency, while only those peers suffering from a failure would have to bear longer restore times.

Finally, our results cover in detail data loss events. We find that such events are practically negligible for a mature P2P application in which permanent host failures dominate peer deaths. We also show the limitations of our technique for a system characterized by a high application-level churn, which is typical of new P2P applications that must conquer user trust.

4.3 A Peer-Assisted Architecture

Currently, a widely used approach to the design of applications for on-line data storage heavily relies on cloud storage systems (e.g., Dropbox [5]) which are used to transparently synchronize, when machines are connected to the Internet, the local copy of data with a remote one residing in a data-center. The success of this approach is undeniable: Dropbox passed in 2012 the milestone of 20 million registered users. However, even if they are undeniably useful, these applications are not free from shortcomings. User data is outsourced to a single company, raising issues about data confidentiality and risk of data loss (the case of Carbonite is emblematic [104]); indeed, companies offering a storage service do not generally offer formal guarantees about their data availability and reliability.

The most significant limitation of current on-line backup applications, though, is cost: bandwidth and storage are expensive, resulting in companies not being able to offer *for free* more than few gigabytes of storage space. This trend will reasonably hold in the future since data-center costs are largely due to energy (power and cooling) and personnel costs rather than hardware costs [12].[4]

In this scenario, it is tempting to think that a pure peer-to-peer architecture would be an ideal solution to eliminate the costs of a cloud-based backup application. However, our work reveals that there are frequent cases in which the resources that peers contribute to the system are simply not sufficient to guarantee that all users will be able to complete their backups in a reasonable amount of time, if ever.

In this Section, we make the case for an hybrid approach that we call peer-assisted: storage resources contributed by peers and sold by data-centers coexist. We focus on two key elements of such a system, *data placement* and *bandwidth allocation*, and study their impact on performance measured by the time required to complete a backup and a restore operation and the end-users' costs.

The main contributions of this work are summarized in the following.

- We show that, by using adequate bandwidth allocation policies in which storage space at a cloud provider is only used temporarily, a peer-assisted backup application can achieve performance comparable to traditional client-server architectures with substantial cost savings.

- We explore the impact of data placement policies on system performance and fairness, and conclude that pure peer-to-peer systems may work only in particular settings and that fairness (in terms of resources obtained and contributed to the system) has a price that we measure by the monetary cost supported by end-users.

- We evaluate the effects of skewed storage demand and resource contribution,

[4]As we will see in the Summary, a recent measurement work we did on a well-known on-line storage system indicates that such costs are instead rapidly sinking. This has important consequences on the design of storage applications, which are now all shifting to a client-server approach.

and conclude that the system architecture we propose copes well with peer heterogeneity.

- We evaluate the effects of the system scale and show that a peer-assisted backup application imposes a limited load on storage servers even when the number of peers in the system grows.

- We show that state-of-the art coding techniques used to ensure data availability at any point in a peer's life-time impose high data redundancy factors, which can be lowered without affecting in a sensible way the ability of peers to restore their data in case of a failure.

4.3.1 System Overview

We now present the design of our peer-assisted backup application, and focus on bandwidth allocation and data placement problems.

Assumptions: In this work we build upon the approach taken by Dropbox [5], and assume users to specify one or more local folders containing important data to backup. We also assume that data selected for backup is available locally to a peer. This is an important trait that distinguishes backup from storage applications, in which data is only stored remotely. As a consequence, *data maintenance*, *i.e.*, making sure that a sufficient number of data fragments are available at any point in time and reacting by generating new fragments when remote peers fail or leave, is greatly simplified.

We assume peers to contribute with non-negligible storage capacity to the system, with ADSL-like bandwidth capacity, and several hours of continuous uptime per day. As we show in our results, nodes contributing with too little resources either exact a high toll in terms of storage capacity of other peers or, when incentive mechanisms are in place, they are not able to sustain by themselves a working system and require the presence of server-based storage.

In this work, we assume the data-center hosting the storage service to offer ideal reliability and availability guarantees and to charge end-users for bandwidth and storage.

Furthermore, we assume the presence of a centralized component, similar in nature to that of the "tracker" in the BitTorrent terminology. The *Tracker*[5] is in charge of membership management, *i.e.*, it maintains a list of peers subscribed to the backup application. Hence, the tracker can bootstrap a new peer with a list of other peers susceptible to store her backup data. The Tracker also implements an additional component used to *monitor* the on-line behavior of a peer: the list of peers in the system is enriched by a measure of the fraction of time a given peer is on-line.

[5]In practice, a tracker can be easily distributed using a DHT approach.

4.3.1.1 Bandwidth Allocation

In this work, we target typical users that connect to the Internet through ADSL: upload bandwidth is a scarce resource that calls for bandwidth allocation policies to optimize its usage. Upload capacity is used to back up local data, for data maintenance and for serving remote requests for data restore.

In our system, a bandwidth scheduler is triggered at regular intervals of time. Restore slots are given the highest priority to ensure that crashed peers are able to recover their data as soon as possible. Backup slots are treated as follows. By default, we employ an *opportunistic* allocation that prioritizes uploads to on-line peers rather to the storage server, with the goal of saving on storage cost. When multiple slots that satisfy this constraint are available, we prioritize pending fragment uploads that are closest to completion. As an alternative, we also study the effects of a *pessimistic* allocation aiming at minimizing the time to backup data: in this case, all the upload slots are devoted to send backup fragments to a storage server.

Since remote peers exhibit an intermittent on-line behavior, our bandwidth allocation aims at completing as soon as possible the transfer of data fragments to remote peers (both in restore and backup operations). Hence, we dedicate the whole[6] capacity to a single upload slot; if a single data transfer does not saturate the upload bandwidth and the backup operation is not finished, the surplus is used to transfer backup fragments to the storage server.

A data backup operation is successful when s fragments have been uploaded to the storage server and $p(s)$ fragments, as defined in Eq. 4.1, have been uploaded to remote peers, ensuring the required target data availability. For example, assume $k = 32$ original blocks of which $s = 15$ are stored on the storage server, and $x = 20$ encoded blocks are currently stored on remote peers. If $x \geq p(s)$, the backup operation is considered successful. Otherwise a new backup fragment is uploaded to a remote peer or to the storage server: in the first case, the number of encoded blocks becomes $x = 21$; in the second case, the number of original blocks stored on the server becomes $s = 16$, resulting in a lower value for $p(s)$. This process continues until x reaches $p(s)$.

Since long-term storage on a server is costly, we introduce an *optimization phase* that begins after a successful backup: peers attempt to offload the storage server by continuing to upload additional encoded blocks to remote peers. Storage servers are used as a *temporary storage* to meet the availability target as soon as possible. Once the number of additional encoded blocks stored on remote peers reaches the $p(s)$ value of Eq. 4.1, the used storage space on the server gets gradually reclaimed. In practice, during the optimization phase a random backup fragment stored on the server is flagged for deletion; subsequently, a peer uploads one encoded block to a remote peer and checks if the number of remotely-stored fragments x is at least $p(s-1)$. If this condition is satisfied, the original block marked for deletion can be

[6]In practice, to use the full nominal rate of the up-link, one must also consider some under-utilization introduced by TCP's congestion control mechanism.

safely removed from the server, otherwise the upload of encoded blocks to remote peers continues until x reaches $p(s-1)$.

Data maintenance operations work as follows: once the local peer detects a remote peer failure, the number of remotely stored fragments x has become lower than $p(s)$. This is equivalent to a situation where the backup is not complete, and handled according to the upload bandwidth allocation policy. When using the opportunistic strategy, a new backup fragment is generated locally and is re-scheduled to be uploaded to a (possibly different) remote peer, or to the remote server if no remote peer is available. With the pessimistic strategy, a new fragment gets transferred immediately to the storage center, and possibly reclaimed afterward during the optimization phase.

Download bandwidth allocation depends on whether data fragments are downloaded from the storage server (*i.e.*, during restore operations) or from remote peers. In our work we assume that restoring one or more fragments from a storage server saturates the down-link of a peer. When data fragments are downloaded from remote peers we avoid over-partitioning the down-link of a peer by imposing a limit on the number of parallel connections: as a consequence the risk of very slow data transfers is mitigated.

4.3.1.2 The Data Placement Problem

Data placement amounts to the problem of selecting the remote location that will store backup fragments. In many P2P storage systems, data fragments are randomly placed on remote peers using a DHT-based mechanism. Since our focus is on backup applications, a look-up infrastructure is not needed. Indeed, a restore operation requires locating a sufficient number of fragments to obtain the original data, hence the only information a peer needs is the list of remote peers currently storing backup fragments. This information is provided by the *Tracker*.

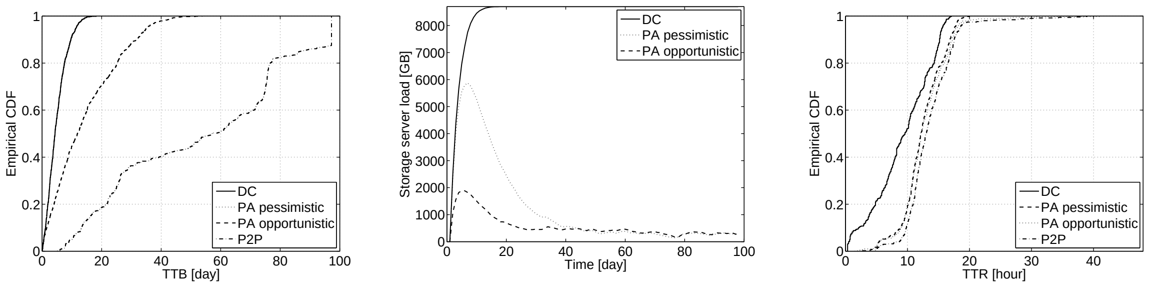
In our work, we study the impact on system performance of two data placement policies:

- *Random*: backup fragments are placed on random remote peers with enough available space and that are not already storing another backup fragment for the same peer.

- *Symmetric Selective*: peers adopt a “tit-for-tat”-like policy and accept to store a (single) fragment for a remote peer only if reciprocity is satisfied. In addition, peers are partitioned into “clusters” depending on their on-line behavior. Peers upload backup fragments exclusively to remote peers within the same cluster.

When the Symmetric Selective policy is used, data availability is computed using a modified Eq. 4.1: the average on-line availability of peers is computed for each cluster. Clustering based on the on-line availability of a peer is performed by the *Tracker*, which constantly monitors the intermittent behavior of the subscribers of the backup application.

4.3.2 Results



(a) TTB. (b) Storage costs. (c) TTR.

Figure 4.3: General overview of the performance of three approaches to on-line backup applications. A peer-assisted design approaches the performance of a server-based solution and achieves small long-term storage costs.

Fig. 4.3 overviews the performance of three alternative approaches to on-line backup applications. We compare a legacy client-server application where users store their data solely on a storage server (labeled DC, which stands for data-center), a P2P application in which the only storage resources available are those contributed by the peers (labeled P2P), and the peer-assisted backup system we discuss in this work (labeled PA). We include both the opportunistic and the pessimistic allocation policy, and use *random data placement*.

Fig. 4.3(a) illustrates the cumulative distribution function (CDF) of TTB values. Clearly, the minimum TTB is achieved by the DC application. Indeed, the redundancy factor is $r = 1$, hence only k backup fragments are uploaded to a central server, which is always available and has infinite inbound bandwidth. Note that also the peer-assisted application with pessimistic allocation (PA-pessimistic) achieves a minimum TTB for the simple reason that, initially, all original backup fragments are uploaded to the storage server (the line for the two cases coincide in Fig. 4.3(a)).

The peer-assisted application with opportunistic allocation (PA-opportunistic) achieves longer times to backup. Indeed, $s+x(s) > k$ backup fragments are uploaded to remote peers to cope with their on-line behavior. Recall that s is the number of original blocks initially uploaded to the central server, before the optimization phase begins.

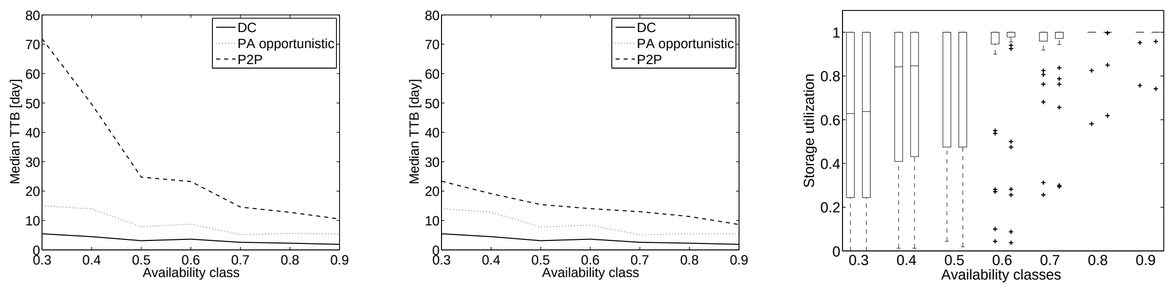
The P2P application obtains the worst results in terms of TTB. More than 15% of peers cannot complete the backup operation, within the simulation time. The very large TTB values are due to two reasons. Firstly, the redundancy factor that meets the target data availability is large and so is the amount of (redundant) backup data to be uploaded to remote peers. Secondly, the intermittent on-line behavior of peers may interrupt data uploads: the bandwidth scheduler enters a time-out phase when no on-line remote peer is available to receive data.

Fig. 4.3(b) depicts the storage costs expressed as a time-series of the aggregate amount of data located in the storage server[7]. The area underlying the curves, multiplied by monetary costs, and normalized by the simulation time, is an indication of the aggregate monetary costs end-users must support.

We observe that in the DC application, the amount of storage requested on the data-center quickly reaches the maximum value. In the PA application, the cost grows as peers upload their fragments to the storage server to complete their backups, and gradually diminishes during the optimization phase. Lower TTB values in the PA-pessimistic case are counterbalanced by higher aggregate costs on the data-center. On the long run, however, the storage load on the server is very low and settles to the same value for both opportunistic and pessimistic allocation. Storage costs do not settle to zero because of the maintenance operations due to peer deaths.

Finally, Fig. 4.3(c) shows the CDF of the TTR metric. We observe that the TTR is much lower than the TTB in all different backup applications. Again, the minimum TTB is achieved in the DC case since the storage server is always on-line and the down-link capacity of a peer is fully utilized. While retrieving data stored on remote peers entails a longer TTR, this quantity remains well within a day in the majority of cases.

Now, we can draw a first important conclusion: by using adequate allocation policies in which a storage server is only used temporarily, a peer-assisted backup application can achieve performance comparable to traditional client-server architectures at much lower costs. Our results also show that a P2P application, despite being free of charge, can meet a reasonable performance only for a small fraction of peers.



(a) TTB with peer deaths. (b) TTB with no peer deaths. (c) Storage load on peers.

Figure 4.4: System performance with random data placement, in terms of median values of TTB and storage load on peers, grouped by availability classes. Note the impact of maintenance traffic due to peer deaths and the uneven distribution of storage load on peers.

Data Placement and Cost of Fairness: We now focus on the impact of the data placement policy on performance, and compare the DC, PA and P2P backup appli-

[7]Storage costs for the P2P application are zero, hence we do not report them on the figure.

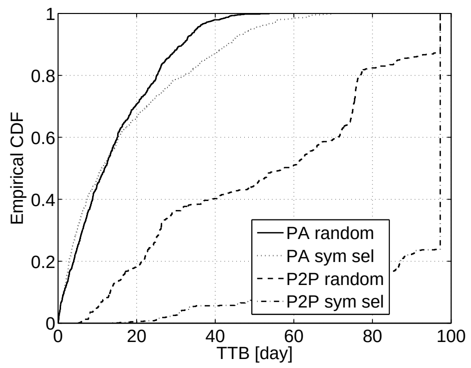
cations. The DC case is shown for reference, since backup fragments are constrained to be stored on the storage server.

First, we study the *random data placement* strategy. Although remote peers are randomly selected, we show results by the availability class of a peer, *i.e.*, peers are grouped depending on their on-line behavior.

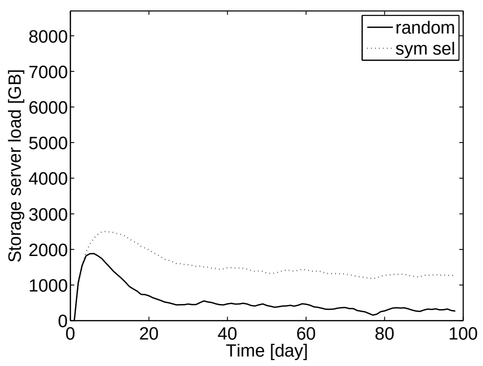
Fig. 4.4(a) and Fig. 4.4(b) represent the median TTB as a function of peer availability class, with and without peer deaths (because of disk crashes) respectively. Clearly, TTB is correlated to the availability class: higher on-line times entail lower TTB. Comparing the case with and without peer deaths reveals the sensitivity of the P2P approach to data maintenance traffic. Instead, the PA application tolerates well peer deaths because the storage server helps in speeding up repairs.

Fig. 4.4(c) shows the amount of data stored by peers: for each availability class, the left box-plot is for the PA case and the right box-plot is for the P2P case. Random data placement introduces unfairness: indeed, peers with a larger on-line time are more likely to be selected as remote locations to store backup fragments and their excess capacity is exploited by peers with low availability. This result motivates the adoption of the *symmetric selective* data placement policy, whose goal is to impose system fairness.

Fig. 4.5(a) shows the CDF of the TTB for PA and P2P applications. For the PA application, the TTB achieved by highly available peers decreases, whereas peers with low availability experience a slightly increased TTB. This trend is more pronounced for P2P applications. Not only the TTB can be very large, but a substantial fraction of peers cannot complete their backup operation: this happens for more than 15% of the cases with a random data placement and for almost 80% of the cases with the symmetric selective policy.



(a) TTB.



(b) Storage costs.

Figure 4.5: Comparison between random and symmetric selective data placement. Introducing fairness has a significant impact on the performance of a P2P design, while it introduces additional costs for a PA application.

Imposing fairness in a PA application modifies the costs for end-users, as illustrated in Fig. 4.5(b). When the symmetric data placement policy is used, peers with low availability cannot exploit the excess capacity in the system and are compelled to store their data on the storage server.

In summary, we justify the system-wide loss in performance with the notion of *cost of fairness*. When peers are constrained to store backup fragments on remote peers with similar on-line behavior and are compelled to offer an amount of local storage space proportional to the amount of (redundant) data they inject in the system, the excess capacity provided by highly available peers cannot be exploited. In a P2P application, peers can suffer a severe loss in performance or eventually cannot complete their backups. Instead, in a PA application system fairness translates into higher storage costs for peers with low-availability, but system performance is only slightly affected.

4.4 Summary

The work presented in this Chapter, and the contributions made to the area of on-line data storage, can be summarized as follows:

- We study the general **problem of scheduling** data transfers in a P2P backup application: essentially, given a series of capacity constraints on bandwidth and storage resources, we seek at finding the right moment to schedule the transfer of information to the right remote machine, given as input the amount of data to be stored and the on-line behavior (that is the availability) of such machines. In Section 4.1 we present our main theoretic result: the scheduling problem can be solved in polynomial time by encoding it in a max-flow formulation. The gist of our approach lays in the subtle construction of a flow (bipartite) network which encode remote machines and the discrete time-slots in which they are available.

  We use the theoretic bounds computed with our model to perform a comparative analysis of on-line heuristics, aiming at exploiting *past* on-line behavior to produce scheduling decisions. Our experiments indicate that this approach is substantially better than random scheduling, yet not more complex to implement.

- We study the problem of **redundancy management** in the context of our P2P backup application. Armed with the realization that backup applications present a particular access pattern, namely "`WRITE` many, `READ` (nearly) never", we design a new on-line mechanism that achieves data durability while requiring a surprisingly low data redundancy. Besides showing that our mechanism is stable (in the "Control Theory" sense), we show that it can be realized in practice by leveraging the properties of rate-less coding.

  Our work is complemented by a series of trace-driven simulations in which peer characteristics in terms of bandwidth and on-line behavior are obtained from real deployed systems. Our results indicate that the price to pay for having lower data redundancy is longer restore operations. Due to the data access patterns we hint at above, we believe such a trade-off to be reasonable: all peers benefit from lower resource utilization and faster backup operations, while only the unfortunate users experiencing a crash experience slightly longer data retrieval operations.

- We present – for the first time – a **peer-assisted architecture** that exhibit a subtle design. Since long term storage for data backup is costly, instead of using Internet storage services as permanent locations where to store data, we use them as temporary buffers that assist peers in the delicate backup operation: data uploads complete as fast as possible; only once data is safe, an optimization phase takes care of removing data from storage servers and placing it on remote peers, for the sake of cost savings.

  We evaluate our system using trace-driven simulations for a wide range of parameters and settings. Our results show that the main mechanism we address – namely bandwidth allocation and data placement – are successful in exploiting server resources: we obtain a system performance that approaches that of a client-server approach, at a small fraction of the costs.

CHAPTER 5

Data Intensive Scalable Computing

Contents

Introduction

This Chapter is dedicated to on-going work, that represents a natural evolution of the line of research presented in the previous Chapters.

The context of this work stems from the realization that the amount of data in our world has been exploding. E-commerce, Internet security and financial applications, billing and customer services – to name a few examples – steadily fuel an exponential growth of large pools of data that can be captured, communicated, aggregated, stored, and analyzed. As companies and organizations go about their business and interact with individuals, they are generating a tremendous amount of digital footprints, *i.e.,* raw, unstructured data –for example log files – that are created as a by-product of other activities.

Nowadays, the ability to store, aggregate, and combine large volumes of data and then use the results to perform deep analysis has become ever more accessible as trends such as Moore's Law in computing, its equivalent in digital storage, and cloud computing continue to lower costs and other technology barriers. However, the means to extract insights from data require remarkable improvements as software and systems to apply increasingly sophisticated mining techniques are still in their infancy. Large-data problems require a distinct approach that sometimes runs counter to traditional models of computing.

In this line of work, that we dub data-intensive scalable computing (DISC), we depart from high-performance computing (HPC) applications and we go beyond traditional techniques developed in the database community. In the sequel of this Chapter we present our current effort devoted to the study and analysis of scalable systems and the design of scalable algorithms to manage, store and process large amounts of data.

For the sake of brevity, in this Chapter we present a small selection of our work. The Chapter is organized as follows:

- In Sect. 5.1 we describe our work towards the design, implementation and performance evaluation of a new scheduling component for the Hadoop MapReduce framework. The aim of this work is to extend the well-known concept of size-based scheduling, whereby the size of a request (or, in our context, a job) is used to take scheduling decisions, such that it can be applied in a parallel setting (hence a multi-server model) in which size information is not available a-priori.
- In Sect. 5.2 we present our work on high-level abstractions of the MapReduce programming model. Specifically, we start by focusing at lower layers of the execution stack of a parallel framework and introduce a novel design pattern to specify MapReduce jobs operating over hierarchical data. In doing so, we focus on a specific application scenario, namely network trace analysis. Then, we build upon the new design pattern and work on several modules of a well-known high-level programming language for Hadoop, namely the Pig system. Our ultimate goal is to perform hierarchical aggregates of data, which are expressed using a compact high-level program, and that result (after compilation) in a single MapReduce job, as opposed to the default behavior of Pig, which generates multiple jobs, each processing data at a different granularity.
- Finally, in Sect. 5.3, we summarize our contributions and briefly mention related work that we omit from this Chapter for the sake of brevity.

5.1 Size-based Scheduling for DISC

The advent of large-scale data analytics, fostered by parallel processing frameworks such as MapReduce [43], has created the need to organize and manage the resources of clusters of computers that operate in a shared, multi-tenant environment. For example, within the same company, many users *share* the same cluster because this avoids redundancy (both in physical deployments and in data storage) and may represent enormous cost savings. Initially designed for few and very large batch processing jobs, data-intensive scalable computing frameworks such as MapReduce are nowadays used by many companies for production, recurrent and even experimental data analysis jobs. This is substantiated by recent studies [36, 90] that analyze a variety of production-level workloads (both in the industry and in academia). A first, important, characteristic that emerges from such works is that there exists a stringent need for *interactivity*. Current workloads include a non-negligible number of small jobs: these are preliminary data analysis tasks involving a human in the loop, which for example seeks at tuning algorithm parameters with a trial-and-error process, or even small jobs that are part of orchestration frameworks whose goal is to launch other jobs according to a work-flow schedule. Furthermore, the studies cited above also unveil the raise of deployment best practices in which – to accommodate both long-lasting and short jobs – Hadoop clusters tend to be over-dimensioned, with an evident loss of efficiency because a large fraction of resources remain idle for long periods of time.

In this work, we study the problem of job *scheduling*, that is how to allocate the resources of a cluster to a number of concurrent jobs submitted by the users, and focus on the open-source implementation of MapReduce, namely Hadoop [6]. In addition to the default, first-in-first-out (FIFO) scheduler implemented in Hadoop, recently, several alternatives [31, 55, 64, 92, 106, 108] have been proposed to enhance scheduling: in general, existing approaches aim at two key objectives, namely *fairness* among jobs and *performance*. Our key observation is that fairness and performance are non-conflicting goals, hence there is no reason to focus solely on one or the other objective. In addition, current best practices also hint at complex engineering constructs by defining separate scheduling queues for jobs with different characteristics, whereby defining the number of queues, their priorities and the amount of resources dedicated to each queue remains a tedious manual exercise.

Here, we revisit the notion of scheduling performance and propose to focus on job *sojourn time*, which measures the time a job spends in the system waiting to be served and its execution time. Short sojourn times cater to the interactivity requirements discussed above. We thus proceed with the design and implementation of a new scheduling protocol that caters *both* to a fair *and* efficient utilization of the cluster resources. Our solution belongs to the category of *size-based scheduling* disciplines, and implements a generic framework that accommodates a variety of *job aging* primitives, which overcome the limitation of the well-known shortest-remaining-processing-time (SRPT) discipline. In addition to addressing the problem of scheduling jobs with a complex structure in a multi-processor system, we propose an efficient method to implement size-based scheduling when job size is not known a-priori. Essentially, HFSP allocates cluster resources such that job size information is inferred while the job makes progress toward its completion. Furthermore, our scheduling discipline supports a variety of preemption mechanisms, including the standard approach of "killing" jobs, waiting for jobs to finish, and a new technique that enables our scheduler to suspend and eventually resume running jobs.

The contribution of our work can be summarized as follows:

- We design and implement the system architecture of a new **size-based scheduler** for Hadoop, including a (pluggable) component to estimate job sizes, a dynamic resource allocation mechanism that strives at efficient cluster utilization and a new set of low-level primitives that complement existing preemption mechanisms. We label our scheduler HFSP, and the complete source code is available as an open-source project.

- We design and implement a new component that brings the well-known concept of "virtual time" for size-based scheduling in the context of a multi-processor system. We call this component **virtual cluster**, and design and implement a new **job aging mechanism** that account for the multi-processor nature of our system.

- We perform an extensive experimental campaign, where we compare the HFSP scheduler to the two main schedulers used in production-level Hadoop deployments, namely the FIFO and the Fair schedulers. For the experiments, we use state-of-the-art workload suite generators that take as input realistic workload traces. In our experiment, we use a large cluster deployed in Amazon EC2.

5.1.1 Hadoop Fair Sojourn Protocol

HFSP is a generic implementation of a size-based preemptive scheduler for Hadoop, which can accommodate a variety of disciplines. Concretely, we embrace a simple abstract[1] idea: prioritize jobs according to the completion time they would have using processor sharing, and use preemption to allocate resources to the highest-priority job. The size of new jobs is at first estimated roughly based on their number of tasks; by observing the running time of the first few tasks, that estimate is then updated.

Although the abstract idea is reasonably simple, in the following we address a variety of issues related to its materialization in a fully functional prototype.

5.1.1.1 The Job Scheduler

HFSP schedules MapReduce jobs, and is intended for the Hadoop implementation of Mapreduce. In this manuscript, we gloss over a detailed description of Hadoop Mapreduce. For background information on Hadoop, and in particular on the design of Hadoop schedulers, the reader can refer to [87]. We note that MapReduce jobs that may require a different number of MAP and REDUCE tasks: hence, a job might need only a fraction of the resources of a cluster, and therefore two or more jobs may be scheduled at the same time.

Job Dependencies. In MapReduce, a job is composed by a MAP phase followed optionally by a REDUCE phase. We estimate job size by observing the time needed to compute the first few "training" tasks of each phase; for this reason we cannot estimate the length of the REDUCE phase when scheduling MAP tasks. Because of this, for the purpose of scheduling choices we consider MAP and REDUCE phases as two separate jobs. For ease of exposition, we will thus refer to both MAP and REDUCE phases as "jobs" in the remainder of this Section. As we experimentally show in Section 5.1.2, the good properties of size-based scheduling ensure shorter mean response time for both the MAP and the REDUCE phase, resulting of course in better response times overall.

[1]Friedman and Henderson [51] originally described a similar approach for a single-queue server system, but never implemented it in practice.

Scheduling Policy. HFSP works by estimating the size of each job using the training module described in Section 5.1.1.2; this size is then used to compute the completion time of each job in a simulated processor-sharing scheduler (see Section 5.1.1.3). When jobs arrive, all cluster resources are assigned to fulfill demands of jobs ranked by increasing simulated completion time. If a job ends up having more running tasks than assigned, excess slots are released by using a preemption mechanism (Section 5.1.1.4 illustrates a novel approach to preemption that complements existing techniques).

Training Phase. Initially, the training module provides a rough estimate for the size of new jobs (Section **??**). This estimate is then updated after the first s "sample" tasks of a job are executed (Sections **??** and **??**). To guarantee that job size estimates quickly converge to more accurate values, the first s tasks of each jobs are prioritized, and the above mechanism is amended so that a configurable number of MAP and REDUCE "training" slots are always available, if needed, for the sample tasks. This number is limited in order to avoid starvation for "regular" jobs in case of a bursty arrival of a large number of jobs.

Data locality. In order to minimize data transfer latencies and network congestion, it is important to make sure that MAP tasks work on data available locally. For this reason, HFSP uses the *delay scheduling* strategy [108], which postpones scheduling tasks operating on non-local data for a fixed amount of attempts; in those cases, tasks of jobs with lower priority are scheduled instead.

5.1.1.2 Job Size Estimation

Size-based scheduling strategies require knowing the size of jobs. In Hadoop, we do not have a perfect knowledge of the size of a job until it is finished; however, we can, at first, estimate job size based on characteristics known *a priori* such as the number of tasks; after the first sample tasks have executed, the estimation can be updated based on their running time.

This estimation component has been designed to result in minimized response time rather than coming up with perfectly accurate estimates of job length; this is the reason why sample jobs should not be too many (our default is $s = 5$), and they are scheduled quickly. We stress that the computation performed by the sample tasks is *not* thrown away: the results computed by sample tasks will be used in the rest of the job exactly as the other regular tasks.

Further details on job size estimation are given in [87].

5.1.1.3 Virtual Cluster

The estimated job size is expressed in a "serialized form", that is the sum of runtimes of each of its tasks. This is useful because the physical configuration of the cluster does not influence the estimated size; moreover, in case of failures, the number of available TaskTrackers varies, but the size of jobs does not change. The job scheduler, though, requires an estimated completion time that depends on the physical configuration of the real cluster. We obtain that by simulating a processor sharing discipline applied on a *virtual cluster* having the same number of machines and the same configuration of slots per machine as the real cluster. The projected finish time obtained in the virtual cluster is then fed to the job scheduler, which will use it to perform its scheduling choices.

Virtual cluster resources need to be allocated following the principle of a fair queuing discipline. Since jobs may require less than their fair share, in HFSP, resource allocation in the virtual cluster uses a *max-min fairness* discipline. Max-min fairness is achieved using a round-robin mechanism that starts allocating virtual cluster resources to small jobs (in terms of their number of tasks). As such, small jobs are implicitly given priority in the virtual cluster, which reinforces the idea of scheduling small jobs as soon as possible.

The HFSP algorithm keeps track of, in the virtual cluster, the amount of work done by each job in the system. Each job arrival or task completion triggers a call to the job aging function, which uses the time difference between two consecutive events as a basis to distribute progress among each job currently scheduled in the virtual cluster. In practice, each running task in the virtual cluster makes a progress corresponding to the above time interval. Hence, the "serialized" representation of the remaining amount of work for the job is updated by subtracting the sum of the progress of all its running tasks in the virtual cluster.

5.1.1.4 Job Preemption

The HFSP scheduler uses preemption: a new job can suspend tasks of a running job, which are then resumed when resources become available. However, traditional preemption primitives are not readily available in Hadoop. The most commonly used technique to implement preemption in Hadoop is "killing" tasks or entire jobs. Clearly, this is not optimal, because it wastes work, including CPU and I/O. Alternatively, it is possible to Wait for a running task to complete, as done by Zaharia *et al.* [108]. If the runtime of the task is small, then the waiting time is limited, which makes Wait appealing. In fact, we suspend jobs using the Waitprimitive for Maptasks which are generally small. However, for tasks with long runtime, the delay introduced by this approach may be too high.

Since Reducetasks may be orders of magnitude longer than Maptasks, to preempt Reducejobs we adopt a more traditional approach, which we name *eager preemption*: tasks are suspended in favor of other jobs, and resumed only when their job is later awarded resources. Eager preemption requires implementing Suspend and Resume primitives: we delegate them to the operating system (OS). Mapand Reducetasks are launched by the TaskTrackeras child Java Virtual Machines (JVMs); child JVMs are effectively processes, which we suspend and resume using standard POSIX signals: `SIGSTOP` and `SIGCONT`. When HFSP suspends a task, the underlying OS proceeds with its materialization on the secondary stor-

	% of Jobs	# Maps	# Reduces	Label
FB09	(Small jobs) 53	≤ 2	0	select
	(Medium jobs) 41	$2 < x \leq 500$	≤ 500	aggregate
	(Big jobs) 6	≥ 500	≥ 500	transform
FB10	39	≤ 1500	≤ 10	expand
	16	≤ 1500	$10 < x \leq 100$	expand and transform
	11	≤ 1500	$x > 100$	transform
	10	$1500 < x \leq 2500$	$x \leq 100$	aggregate
	7	$1500 < x \leq 2500$	$x > 100$	transform
	10	$x > 2500$	$x > 100$	transform

Table 5.1: Job types in each workload, as generated by SWIM. Jobs are labeled according to the analysis tasks they perform. For the `FB2009` dataset, jobs are clustered in bins and labeled according to their size.

age (in the swap partition), *if and when* its memory is needed by another process. We note that our implementation introduces a new set of states associated to an Hadoop task, plus the related messages for the JobTracker and TaskTracker to communicate state changes and their synchronization.

There are many more considerations that need to be discussed in detail for a proper handling of eager preemption. For the sake of brevity, we gloss over such details and refer the reader to [87].

5.1.2 Experiments

This Section focuses on a comparative analysis between FAIR and HFSP schedulers. Next, we specify the experimental setup and present a series of results, that we call macro benchmarks. Macro benchmarks illustrate the global performance of the schedulers we study in this work, in terms of job sojourn times. Additional results are available in a technical report [87].

5.1.2.1 Experimental Setup

In this work we use a cluster deployed on Amazon EC2 [1], which we label the Amazon Cluster. The Amazon Cluster is configured as follows: we deploy 100 "`m1.xlarge`" EC2 instances, each with four 2 GHz cores (eight virtual cores), 4 disks that provide roughly 1.6 TB of space, and 15 GB of RAM.[2] In our experiments the HDFS block size is set to 128 MB and a replication factor of 3, while the main Hadoop configuration parameters are as follows: we set 4 Map slots and 2 Reduce slots per node.

[2]This is the configuration used by Zaharia *et. al.* [108].

Workloads. Generating realistic MapReduce workloads is a difficult task, that has only recently received some attention. In this work, we use SWIM [?], a standard benchmarking suite that comprises workload and data generation tools, as described in the literature [35–37]. A workload expresses in a concise manner *i)* job inter-arrival times, *ii)* a number of Map and Reduce tasks per job, and *iii)* job characteristics, including the ratio between output and input data for Map tasks. For our experiments, we use two workloads synthesized from traces collected at Facebook and publicly available [37] – that we label `FB2009` and `FB2010` – as done in previous works [37, 108]. Table 5.1.2 describes the workloads we use, as generated by SWIM for a cluster of 100 nodes.

The `FB2009` workload comprises 100 unique jobs, and is dominated by small jobs, i.e., jobs that have a small number of Map tasks and no Reduce tasks. The job inter-arrival time is a random variable with an exponential distribution and a mean of 13 seconds, making the total submission schedule 22 minutes long. Experiments with the `FB2009` dataset illustrate scheduling performance when the system is not heavily loaded, but has to cope with the demand for interactivity (small jobs).

The `FB2010` workload comprises 93 unique jobs, which have been selected by instructing SWIM to filter out small jobs. In particular, the number of Map tasks is substantially larger than the number of available slots in the system: Map phases require multiple "waves" to complete. The number of Reduce tasks varies between 10 and the number of available reduce slots in the cluster. The job inter-arrival time is a random variable with an exponential distribution and a mean of 38 seconds, making the total submission schedule roughly 1 hour long. The `FB2010` dataset is particularly demanding in terms of resources: as such, scheduling decisions play a crucial role in terms of system response times.

Scheduler Configuration. Unless otherwise stated, HFSP operates with the delay scheduling technique and eager preemption enabled. HFSP requires a handful of parameters to be configured, which mainly govern the estimator component: we use a uniform distribution to perform the fitting of the estimate job size; the sample set size s for Map and Reduce tasks is set to 5; the parameter Δ to estimate Reduce task size, is set to 60 seconds; we set the parameter $\xi = 1$. For the workloads we use in our experiments, the parameters described above give the best results.

The FAIR scheduler has been configured with a single job pool, using the parameters suggested in the Hadoop scheduler documentation.

5.1.2.2 Macro Benchmarks

Using the Amazon Cluster, we now report the empirical cumulative distribution function (CDF) of sojourn times for FAIR and HFSP, when the cluster executes the `FB2009` and `FB2010` workloads. Although we do not include results we obtain with the FIFO scheduler, for reference, our experiments report a mean sojourn time 5 to 10 times bigger than that of HFSP, depending on the workload.

Figure 5.1 groups results according to job sizes: the `FB2009` dataset contains

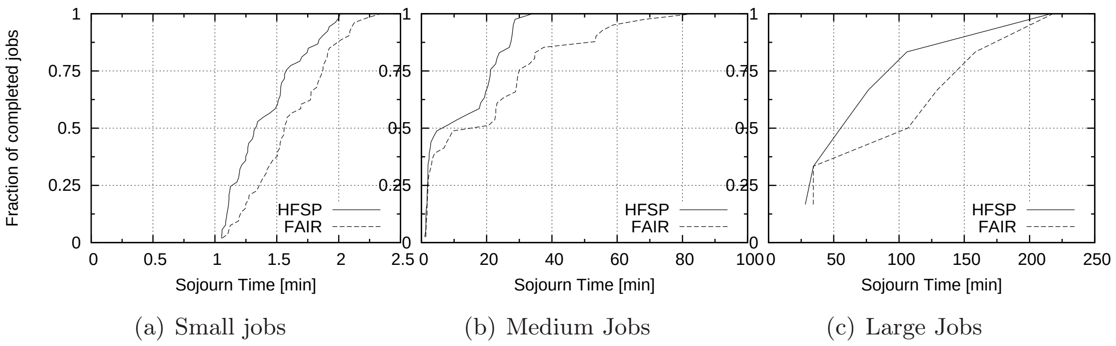


Figure 5.1: ECDFs of sojourn times for the FB2009 dataset. Jobs are clustered in classes, based on their sizes. HFSP improves the sojourn times in most cases. In particular, for small jobs, HFSP is slightly better than FAIR, whereas for larger jobs, sojourn times are significantly shorter for HFSP than for FAIR.

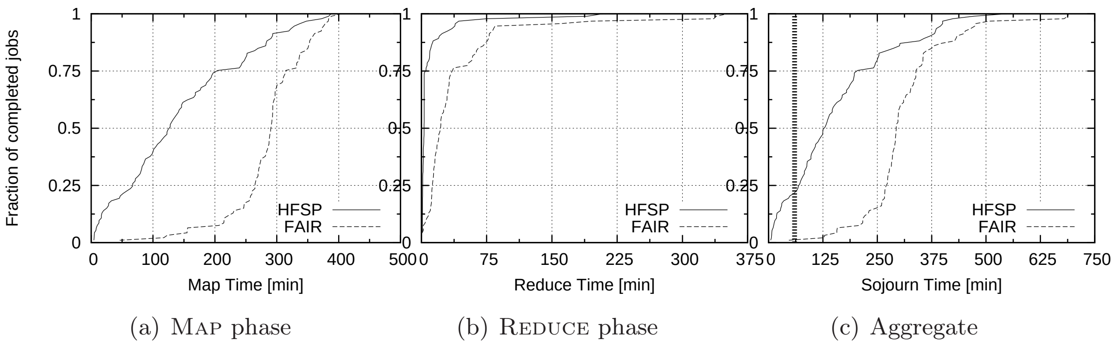


Figure 5.2: ECDFs of sojourn times for the FB2010 dataset. The Map phase (left) shows significant improvements of HFSP over FAIR. Shorter sojourn times in the Map phase are reflected in the Reduce phase (middle), which shows that HFSP outperforms FAIR in terms of sojourn times. The ECDF of aggregate job sojourn times (right) indicates that both median and worst-case sojourn times are better with HFSP. The vertical line at 60 minutes (the workload has been consumed) indicates 20% of completed jobs with HFSP vs. 1% of completed jobs with FAIR.

three distinct clusters of job sizes (small, medium and large), with small jobs dominating the workload. Our results indicate a general improvement of job sojourn times in favor of HFSP. For small jobs, the fair share of cluster resources allocated by both HFSP and FAIR is greater than their requirements in terms of number of tasks. In addition, very small jobs (with 1-2 Map tasks only) are scheduled as soon as a slot becomes free (both under the HFSP and FAIR), and therefore their sojourn time depends mainly on the frequency at which slots free-up and on the cluster state upon job arrival. Overall, we observe that HFSP performs slightly better than FAIR for small jobs. For medium and large jobs, instead, an individual job may require a significant amount of cluster resources. Thus, the advantage of HFSP is mainly due to its ability to "focus" cluster resources – as opposed to "splitting" them according to FAIR – towards the completion of the smallest job waiting to be served.

Figure 5.2 shows the results for the `FB2010` dataset. We show the sojourn times of the Map and Reduce phases, and the total job sojourn times. In the Map phase (Figure 5.2(a)), the sojourn times are smaller for HFSP than for FAIR: the median sojourn time is more than halved. This is a consequence of the HFSP discipline, which dedicates cluster resources to individual jobs rather than spreading them on multiple "waves". As such, with HFSP, Reduce tasks enter the Shuffle phase earlier than what happens with FAIR, and have to wait less time for all their input data to be available. Therefore, the Reduce phase is shorter with HFSP, as shown in Figure 5.2(b). Clearly, the HFSP discipline in the Reduce phase also contributes to short sojourn times, with a median difference of roughly 30 minutes. Finally, the total job sojourn times indicate that the system response time with HFSP substantially improves. For illustrative purposes, we show a vertical line corresponding to 60 minutes, by which all jobs of the `FB2010` workload arrived in the system. By that time, only one job completes with FAIR, whereas more than 20% of the jobs, are served with HFSP. More to the point, when 80% of jobs are served by HFSP, roughly 15% of jobs are served with FAIR.

In summary, HFSP caters both to workloads geared towards "interactive" small jobs and to a more efficient allocation of cluster resources, which is beneficial to large jobs.

The HFSP scheduler we proposed in this work brought up several challenges. First, we came up with a general architecture to realize *practical* size-based scheduling disciplines, where job size is not assumed to be known a priori. The HFSP scheduling algorithm solved many problems related to the underlying discrete nature of cluster resources, how to keep track of jobs making progress towards their completion, and how to implement strict preemption primitives. Then, we used standard statistical tools to infer task time distributions and came up with an approach aiming at avoiding wasting cluster resources while estimating job sizes. Finally, we performed a comparative analysis of HFSP with two standard schedulers that are mostly used today in production-level Hadoop deployments, and showed that HFSP brings several benefits in terms of shorter sojourn-times, even in small, highly utilized clusters.

5.2 High-level Languages for DISC

As evidence of the current trends in state-of-the-art methods to perform data analysis at a very large scale, programmers have been flocking to the procedural MapReduce programming model [43]. A MapReduce program essentially performs a group-by-aggregation in parallel over a cluster of machines. The programmer provides a map function that dictates how the grouping is performed, and a reduce function that performs the aggregation. What is appealing to programmers about this model is that there are only two high-level declarative primitives (map and reduce) to enable parallel processing, but the rest of the code, *i.e.*, the map and reduce functions, can be written in any programming language of choice, and without worrying about parallelism.

Unfortunately, the MapReduce model (and its physical open-source incarnation, Hadoop) has its own set of limitations. Its one-input, two-stage data flow is extremely rigid. To perform tasks having a different (or complex) data flow, programmers have to, in addition to specify their algorithms, understand and modify internal modules of Hadoop and devise composite keys to “route" data across parallel machines. This approach often relies on *design patterns*, that are adapted individually to each analytic task. Also, custom code has to be written for even the most common operations, e.g., projection and filtering. These factors lead to code that is difficult to reuse and maintain, and in which the semantics of the analysis task are obscured. Moreover, the opaque nature of the map and reduce functions impedes the ability of the system to perform optimizations.

Apache Pig [85], Apache Hive [2] and Google Tenzing [32] are representative examples of systems supporting an SQL-like abstraction on top of the MapReduce layer. In general, these systems support basic operators and build upon the huge literature on traditional database systems to optimize the execution plan of the queries, as well as their translation into MapReduce programs. Apart from Tenzing, which is the most complete yet closed-source solution, Pig and Hive have limited optimization capabilities, that only cover the manner in which complex queries are translated into multiple MapReduce jobs. Cascading [3], Stratosphere [10] and others offer an alternative way – not based on SQL – to produce MapReduce programs. Cascading is an abstraction layer aiming at complex operations on data, which require chained MapReduce jobs. It uses a model based on the “pipes and filters" metaphor: an appropriate combination of such operators produces a parallel program that is executed in Hadoop. Similarly, Stratosphere introduces the PACT programming model [15] and a new execution framework called Nephele [103], which replaces Hadoop. The PACT programming model augments MapReduce with new operators to simplify the design of parallel algorithms. Finally, ScalOps builds upon the Hyracks [23] parallel-database and offer a domain- specific language specialized in machine learning.

In this Section we present our on-going work to introduce an important optimization to the Pig system and its Pig Latin scripting language. We start by describing an application scenario involving the analysis of network data, *i.e.*, packet dumps of traffic flowing in a computer network. In doing so, we present a number of data analysis tasks (jobs) and express them as (a sequence of) MapReduce programs. Such programs (which we also call jobs) have been optimized and the technique to achieve efficient job running time has been generalized to come up with a new design pattern, that we label “in-reducer grouping". In-reducer grouping essentially implements a widely used SQL operator, namely the `ROLL UP` operator.

We conclude with an overview of our current work aiming at integrating our design pattern in the Apache Pig internal engine.

5.2.1 Hierarchical data analysis

The endeavor of this Section is to overview our work towards the design of scalable and efficient algorithms for a specific application context, namely network data analysis. In such application, the goal is to analyze a vast amount of data collected from one or more vantage points in the network and stored in a centralized location. Network data is historical in nature, which implies a simple *write once, read many* workload: traces are timestamped and, once a dataset is produced, no updates are required. Applications of network data analysis include anomaly detection, traffic classification, botnet detection, quality of service monitoring and many more. In a typical setting, network data are periodically accumulated in a storage system for future processing and analysis. Therefore, data import operations are required to complete in a time that is strictly less than the accumulation period. The subsequent processing phase, especially when the volume of data to analyze is large, may not be feasible with traditional network analysis tools [79]: in this work, we show how these tasks can be achieved with the Hadoop system and the MapReduce programming model.

We proceed by first defining a set of data analysis jobs, which are detailed in Section 5.2.1.1. Our jobs are representatives of typical (from a Network Operator perspective) network analysis tasks related to users' and protocol behavior. They are run on a large, publicly available network trace from a trans-Pacific Internet link. The aim of our experiments is to describe and discuss the large number of "knobs" that can be tuned to achieve reasonable system performance and to illustrate, using our sample jobs, that the design and implementation of such jobs is far from being trivial and requires great zeal.

Our main contribution is a novel technique for analyzing hierarchical data, and in particular time-stamped logs. Our approach produces a compact MapReduce execution plan for jobs that otherwise would require a complex and lengthy sequence of intermediate jobs. A detailed discussion and a performance evaluation of an alternative approach, based on a distributed database, to execute the jobs outlined in this Section is available in [14].

5.2.1.1 Datasets, jobs, and their implementation

In this Section we briefly describe the analyzed datasets, and provide a high-level description of the four sample jobs. We then describe the design of such jobs following the MapReduce programming model.

Network Data: The input to our Jobs is a textual CSV file of network traces from the publicly available MAWI (Measurement and Analysis on the WIDE Internet) archive, captured from a trans-Pacific link between the United Stated and Japan [38]. In particular, we use traces from sample-point F collected from 2009/03/30 to 2009/04/02. Packet payload is omitted and IP addresses are anonymized in the original trace. From it we extracted a CSV file consisting of one record per packet. Each record has a size of approximately 64 B and includes information like time-stamp, source and destination IP addresses and ports, flags, used protocol and packet length. The original trace consists of 432 GB raw data, resulting in a CSV file of roughly 100 GB.

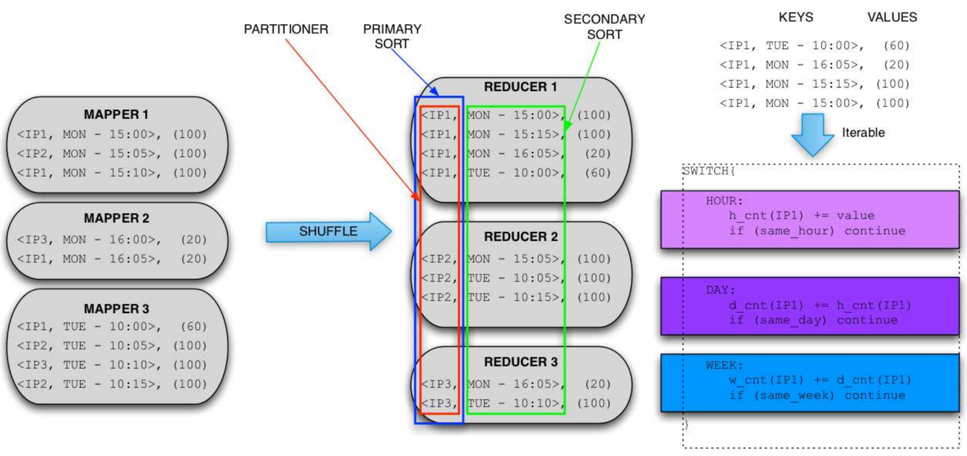
Sample Jobs: Hereafter we describe four sample jobs representative of typical analysis tasks run by network operators. Each job compute statistics hourly, daily and for the whole trace duration.

- **User Ranking (J1).** For every IP address, compute the number of uploaded, downloaded and total exchanged (upload + download) bytes.
- **Frequent IP/Port by Heavy Users (J2).** Heavy users are defined as the top-10 IP addresses from Job J1 in terms of total exchanged bytes. This Jobs computes the top IP/Port pairs for all heavy users.
- **Service Ranking (J3).** For experimental purpose, the set of IP addresses is split arbitrarily into two subsets, $\mathcal{A}$ (servers) and $\mathcal{B}$ (clients). For every IP address in $\mathcal{A}$, this Jobs computes the total number of connections established by IP addresses in $\mathcal{B}$.
- **Unanswered TCP Flows (J4).** For every IP address, this Job computes the number of unanswered TCP flows. A TCP flow is considered *answered* if the initial SYN packet is acknowledged by a SYN-ACK within 3 seconds.

Job Implementation in MapReduce. MapReduce jobs require to define a Map and a Reduce function, the "routing" of data from mappers to reducers (called Partitioning and Grouping), the order of data received by the reducers and possibly, to further optimize the job, a Combiner. We remark that there are several ways to implement our sample Jobs. Although we have tried many alternatives, in the following we only present best-performing implementations.

Before proceeding any further, we discuss the key idea underlying the conceptual design of our jobs. As an illustrative example, let's consider the fact that our jobs compute statistics at different time granularities, namely hours, days and weeks A naive approach is to launch a Job for each time period and, if possible, use the output of Jobs executed on fine grained periods as input for Jobs running on coarse grained periods. However, this approach requires various cycles of data materialization and subsequent de-materialization (not to mention framework overhead), that can be avoided by a careful design of the Reduce, Grouping, Sorting and Partitioning phases.

In this work we present a new "design pattern" that we labeled *in-reducer grouping*, whose aim is to produce a compact execution plan instead of using a sequence of jobs. To understand our technique, it is convenient to recall the purpose of the standard Grouping phase in Hadoop. Grouping is used to prepare the input data to the Reduce phase by grouping all intermediate key/value pairs in output from the Map phase that refer to the *same* key. As such, the Reduce phase receives a


PARTITIONER
PRIMARY SORT
SECONDARY SORT
MAPPER 1
<IP1, MON - 15:00>, (100)
<IP2, MON - 15:05>, (100)
<IP1, MON - 15:10>, (100)
MAPPER 2
<IP3, MON - 16:00>, (20)
<IP1, MON - 16:05>, (20)
MAPPER 3
<IP1, TUE - 10:00>, (60)
<IP2, TUE - 10:05>, (100)
<IP3, TUE - 10:10>, (100)
<IP2, TUE - 10:15>, (100)
SHUFFLE
REDUCER 1
<IP1, MON - 15:00>, (100)
<IP1, MON - 15:15>, (100)
<IP1, MON - 16:05>, (20)
<IP1, TUE - 10:00>, (60)
REDUCER 2
<IP2, MON - 15:05>, (100)
<IP2, TUE - 10:05>, (100)
<IP2, TUE - 10:15>, (100)
REDUCER 3
<IP3, MON - 16:05>, (20)
<IP3, TUE - 10:10>, (100)
KEYS
VALUES
<IP1, TUE - 10:00>, (60)
<IP1, MON - 16:05>, (20)
<IP1, MON - 15:15>, (100)
<IP1, MON - 15:00>, (100)
Iterable
SWITCH{
HOUR: h_cnt(IP1) += value if (same_hour) continue
DAY: d_cnt(IP1) += h_cnt(IP1) if (same_day) continue
WEEK: w_cnt(IP1) += d_cnt(IP1) if (same_week) continue
}

(a) MAP, SHUFFLE, REDUCE

(b) Pipelined computation

Figure 5.3: An illustration of the *in-reducer grouping* technique.

series of $(key, list < values >)$ for further processing. By default, the GROUPING phase in Hadoop is *implicit* and executed at the framework level.

The gist of our approach is to make the GROUPING phase *explicit*, by moving its definition and execution at the user-code level. Fig. 5.3 gives an high-level overview of the *in-reducer grouping* technique, applied to a simplified version of job J1, for the sake of clarity. Figure 5.3(a) represents three machines executing the MAP and REDUCE tasks. The input of each MAP task is in the form of `(key, value)` pairs, where the composite `<key>` is the concatenation of a (source) IP address and a time-stamp, and the `value` refers to the amount of bytes uploaded by the IP address in the `key`. The SHUFFLE phase implements a custom PARTITIONING that uses only the IP address of the composite key defined above, which is also used for primary SORTING; a secondary SORTING takes care of ordering the input to REDUCE tasks based on the second part of the key, namely the time-stamp. In summary, intermediate data from MAP tasks to REDUCE tasks is “routed" based on the IP address, but carries information based on the fine-grained time-stamp.

Now, Figure 5.3(b) illustrates what happens in REDUCE tasks: input data (which technically is an `iterable`) is fed to multiple REDUCE methods that are called based on a condition. The hierarchical nature of data drives how such conditional processing takes place: note that here we assume the function applied to input data to be **algebraic**, *i.e.*, to be associative and distributive. In the example scenario we use for the Figure, we define three conditions based on the time granularities we perform for job J1, that is hours, days and week.

Next, we present (at a fairly low level) the design of each Job described in this Section.

Job J1: For this Job, we need to compute the total bytes sent and received for each IP address per hour/day/whole trace. Therefore, we aggregate data by time-range and IP. In the Map phase, for each record, we emit two records with a composed key `<time-stamp, IP>`. The value accounts for the packet size and the direction (sender or receiver). The Reduce phase receives all the data for each IP ordered by time-stamp, uses a counter for each time-range, and immediately emits the output per time-range.

This Job requires a Partitioning function based on the IP that assigns all the records with the same IP to the same reducer, and a Sorting function that sorts the data based on the IP as the primary key, and the time-stamp as the secondary key.[3] As opposed to a naive approach, this Job reduces scheduling overhead of the framework, disk and network I/O, but is more complex to write.

Job J2: This Job requires two phases: *(J2.1)* find heavy users, and *(J2.2)* find top ports of heavy users.

The input data of Job J2.1 is the output of Job J1. We use an hash table of fixed-size priority queues in the Map phase, that stores only the top 10 users for each input data block. In the Reduce phase, we compute the heavy users over all intermediate data from mappers.

In Job J2.2 we need to read the input CSV file, and emit a record only for heavy users. Hence, we use a distributed cache to save the list of heavy users, so that it is locally available to each Map function. During the setup phase of each mapper, we read the contents of the distributed cache and load them in a hash map. It is important to note that the top-10 ports per day cannot be calculated from the top ports per hour: the "top" operation is not distributive. Job J2.2 is a single Job, where for each record containing an IP from a heavy user, the Map emits three records, using the composite key `<time-range, IP>` and the port and the size as value. The Reduce receives all the records belonging to the same composite key: a fixed size priority queue is used to emit top ports per distinct key. We tested four alternative approaches to implement Job J2.2. In general, we noticed problems related to memory requirements in the Reduce phase and volume of intermediate data transmitted in the Shuffle phase. The approach described above is the one that, in our experiments, performed better.

Job J3: Given two subsets $\mathcal{A}, \mathcal{B}$, we find the total number of unique IPs in $\mathcal{A}$ contacted by IPs in $\mathcal{B}$, as follows. For each record in the input file whose source IP is in $\mathcal{B}$ and whose destination IP is in $\mathcal{A}$, we emit a record containing the IP address in $\mathcal{A}$, the IP address in $\mathcal{B}$ and the time-range which the record falls into.

In the Map phase, we emit one record with a composite key `<IP` $\in \mathcal{A}$ `, IP` $\in \mathcal{B}$`, time-stamp>` for each input record from the CSV file. This choice of composite key simplifies the Reduce function: instead of (naively) using a simple key `<IP` $\in \mathcal{A}$`>`, we exploit the sorting capabilities of the framework to achieve our goal.

[3] An additional detail: we use a hour-based Grouping comparator that simplifies the Reduce function.

The REDUCE function receives all packets belonging to the same composite key and counts the number of unique IP in $\mathcal{B}$. This approach requires a custom PARTITIONING function that operates on IP address in $\in \mathcal{A}$ and a custom SORTING function on the whole composite key.

Among all alternative implementations of this Job we experimented with, this approach is the hardest to code, but uses only a single scan of the input data, and produces a small quantity of intermediate data for the SHUFFLE phase, which also lowers sorting effort. However, it requires more computational resources in the REDUCE phase.

Job J4: To find the unanswered SYNs for each IP address we need to examine the whole dataset, filtering the records with SYN and SYN-ACK flags. This can be easily done in the MAP phase. In order to find if a SYN is unanswered, the REDUCE function receives all the SYNs and all the SYN-ACKs having respectively the same source and destination IP address in the same time-range. The MAP phase behaves as follows. For each SYN packet it emits a record with a composite key `<SrcIP, DstIP, SrcPort, DstPort, time-stamp>`, and SYN as value. For each SYN-ACK packet, it emits an "inverted" record with a composite key `<DstIP, SrcIP, DstPort, SrcPort, time-stamp>`, and SYN-ACK as value. The REDUCE function receives the SYN and all the possible subsequent SYN-ACK in order, so that it can easily check if the SYN-ACK is present.

Using a custom PARTITIONING based on the IPs and ports only, all the packets belonging to the same, bidirectional, flow are sent to the same reducer. Using a custom SORTING function based on the whole key, the REDUCE function receives all the packets belonging to the same bidirectional flow ordered by time-stamp, so that it is easy to check if a SYN is followed by a SYN-ACK.

5.2.2 High-level language support for hierarchical data analysis

The endeavor of this Section is to overview our on-going work to extend and generalize the "in-reducer grouping" design pattern described above. With reference to the application scenario we used previously (and the relative jobs discussed above), computation of aggregates over multi-dimensional data falls into the broader context of On-line Analytical Processing (OLAP).

Thus, if we consider the network data analysis examples defined above, a traditional approach to organize information in a data warehouse system is to use the star schema [33], which splits "dimensions" from "fact" data. Dimension and fact data are stored into separate tables which are referenced via a unique identifier: for every entry in the dimension table there might be one or more entries in the fact table. For network data, which records the traffic flowing on a given link, *dimensions* can be the IPs addresses, ports, protocols used while *fact data* can be time-stamp, flags, packet/byte counters. In the following, we shall use a simplified example scenario for the sake of clarity. We thus consider a data warehouse maintaining sales information as: `<city, state, country, day, month, year, sales>`, where `(city, state,`

Table 5.2: Example sales table.

City	State	Country	Day	Month	Year	Sales
Manhattan	NY	USA	12	12	2010	1000
Los Angeles	CA	USA	02	09	2011	500
Manhattan	NY	USA	1	12	2010	100
Sacramento	CA	USA	14	2	2011	9000
Manhattan	NY	USA	1	1	2011	1000

`country)` are attributes of the location dimension and `(day, month, year)` are attributes of the temporal dimension. Table 5.2 illustrates a sample schema and the corresponding table (in de-normalized form) for the simple example discussed above.

OLAP consists of three basic analytical operations: consolidation (namely `CUBE` and `ROLL-UP`), drill-down, and slicing and dicing. In this work we focus on consolidation operations, which involve the aggregation of data that can be accumulated and computed in one or more dimensions. Cube analysis provides a convenient way to compute aggregate *measures* over all possible groups defined by the two dimensions defined in the simple example above. Note that many of the established cube computation techniques – used in traditional database systems – take advantage of the measure being **algebraic**, a property that allows measures of a super-group to be computed from its sub-groups (e.g. `SUM(a+b) = SUM(a) + SUM(b)`). In traditional database systems, algebraic measures also allow parallel aggregation of data subsets whose results are then post-processed to derive the final result. In the sequel of this section, we will focus solely on algebraic measures.

The focus of our work is to extend CUBE computation techniques to the MapReduce paradigm, by leveraging the "in-reducer grouping" design pattern introduced in the previous section. More to the point, we focus on a high-level language (similar in spirit to traditional SQL) to design data analysis tasks, and extend the recently introduced `CUBE` and `ROLL-UP` operators. Next, we briefly overview PigLatin and Pig, which are respectively the language and the system supporting the language that we use in our work.

Pig and PigLatin. The Pig system (which is an open-source project part of the Hadoop ecosystem, detailed in [53, 85]) takes a Pig Latin program as input, complies it into one or more MapReduce jobs, and then executes those jobs on a given Hadoop cluster. Some of the salient features of the Pig Latin high-level language include: a step-by-step data-flow paradigm to express data analysis, where computation steps are chained together through the use of variables; the use of high-level transformation and operators (e.g. `FILTER`, `GROUP BY`, etc.); the ability to specify an optional schema to annotate the data being processed; and the possibility for the user to define custom functions (UDF) to operate on data.

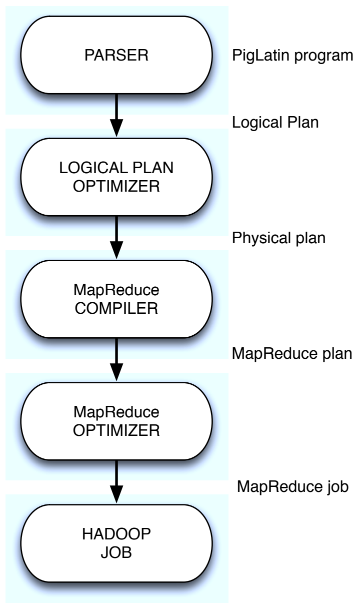


Figure 5.4: Pig compilation and execution stages

A Pig Latin program goes through a series of transformation steps before being executed, as illustrated in Figure 5.4. The first step is *parsing*: the parser verifies that the Pig Latin program is syntactically correct and that all referenced variables are properly defined. The output of the parser is a canonical **logical plan**, that defines a one-to-one correspondence between the Pig Latin statements and the logical operators, which are arranged in a directed acyclic graph (DAG). The logical plan is passed through a *logical optimizer*, which implements many of the traditional optimization techniques developed by the database community, such as projection push-down, early filtering and so on. The logical plan is then transformed into a **physical plan**, which maps logical to physical operators: the physical plan follows the same structure as the logical plan (DAG) but its operators – and their implementation – is close to the MapReduce programming model. Finally, the physical plan is transformed into a **MapReduce plan**, which is compiled and optimized before it is launched for execution. Optimization at the MapReduce plan include, for example, multi-query execution, which avoid reading input files multiple times.[4]

Figure 5.5, illustrates a simple Pig Latin program, and its translation in a Logical, Physical and MapReduce plan. With reference to Table 5.2, let's assume to have a comma separated value file representing the materialization of such table on disk. The simple Pig Latin program we illustrate next computes the total amount of sales per year: first, the code in Algorithm 2, illustrates the source code of the Pig Latin script in charge of performing the computation; then, Figure 5.5 illustrates how the simple program is translated into a logical plan, and what are the corresponding

[4]Recall that MapReduce is generally used for I/O intensive data analysis tasks, hence moving data should be avoided at all costs.

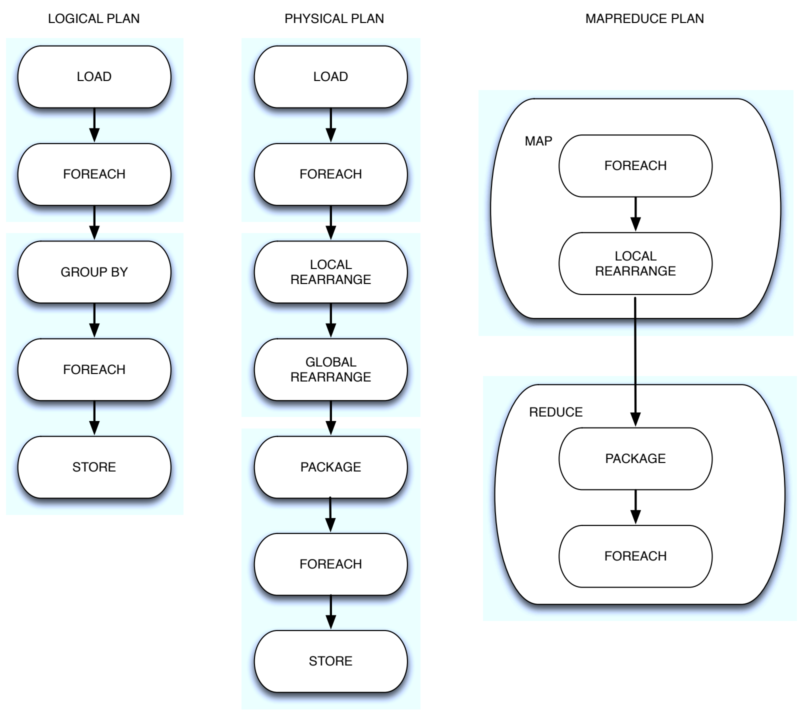


Figure 5.5: Pig program example: logical, physical and MapRedyce plans.

physical and MapReduce plans.

Algorithm 2 A simple Pig Latin program

```
1: A = LOAD './input/data.csv' USING PigStorage(',') AS (city: chararray, state:
   chararray, country: chararray, day: int, month: int, year: int, sales: longint)
2: B = FOREACH A GENERATE year AS y, sales AS s;
3: C = GROUP B BY y;
4: D = FOREACH C GENERATE group, SUM(B.s);
5: STORE D INTO './output/';
```

Using the reference example in Figure 5.5, the most important point to emphasize is that corresponding to the `GROUP BY` operator, which is essentially the operation performed by the SHUFFLE and SORTING mechanism implemented by the MapReduce framework. Indeed, as shown in the figure, the simple Pig Latin program we discuss translates into a single MapReduce job with a MAP and a REDUCE phase.

Next, we glance over the ROLL-UP operator that have been recently added to the Pig Latin language and explain how the “in-reducer grouping" design patterns can be adopted in Pig.

ROLL-UP optimization. We now focus on a widely used consolidation data analysis, namely the `ROLL-UP` aggregate operator. Let’s consider the simple example of a sales table, which is depicted in Table 5.2. Note that the example table is written in a *de-normalized* form: in a data warehouse system, instead, it would be stored as dimensions and fact tables, where dimensions correspond to location and temporal information. We use the de-normalized form here because this is how data is presented and ingested by Pig and the underlying MapReduce system.

Essentially, a typical consolidation analysis would consider the computation of partial aggregates over the location and temporal dimensions, by expressing *grouping sets* over which the `ROLL-UP` would operate. For example, a grouping set of the form `<City, State, County>` generates the following groups for the ROLL-UP operator: `(City, State, County)`, `(*, State, County)`, `(*, *, County)`, `(*, *, *)`. Such groups correspond to computing the aggregate sales at different levels of granularity: per city, state, country and finally totals.

Recent versions of Pig Latin[5] allow the programmer to express the above aggregate computation using a traditional `ROLL-UP` operator. In practice, the Pig parser detects the new operator, and generates a logical plan (that is later transformed into a series of MapReduce jobs) in which a single `ROLL-UP` statement is expanded into a multitude of `GROUP-BY` operators, one for each group in the grouping set defined above. The logical plan is subsequently translated into a physical plan and finally a MapReduce plan, consisting of one job per group, plus a final aggregation step (that cannot be parallelized) to generate the totals.

The contribution of our work is to extend the ability of the Pig system of recognizing optimization opportunities that arise in contexts exemplified by the running example described above. Clearly, the “vanilla" version of Pig is sub-optimal when generating the MapReduce plan for `ROLL-UP` aggregates with algebraic measures: instead of having a number of jobs corresponding to the number of groups, we show next that it is possible to have a much more compact representation of the aggregation operation which translates into a single MapReduce job. Before proceeding any further, we note that (for the purpose of the applications we study in our work), the last aggregation step (computing the total sales) is not required.

Our work consists in modifying the logical plan optimizer component, detect a `ROLL-UP` aggregate, and instruct Pig to generate a physical plan that uses a new operator to perform hierarchical (and algebraic) aggregates on data. The new operator, that we label `H-FOREACH`, implements the “in-reducer grouping" strategy discussed in this section. In practice, the new physical operator works as follows:

- It groups data using as a key[6] a coarse-grained group (country, in the example above)
- It applies the aggregate computation on fine-grained group (the city, in the example above)
- It produces a single MapReduce job implementing a pipelined version of the

[5]At the time of the writing of this manuscript, `CUBE` and `ROLL-UP` aggregates are not yet included in the stable version of Pig.

[6]Recall that the underlying MapReduce programming model operates on key,value pairs

algebraic function. In the example above, it is easy to see that partial aggregates of total sales per City can be used to compute those per State, which in turn can be used to compute those per Country.

In conclusion, our approach allows the computation of algebraic `ROLL-UP` aggregates on multiple groups in a single pass over the whole data, which represent enormous cost savings in terms of I/O operations. Our current research agenda includes an experimental validation and performance evaluation of the benefits of our approach when compared to the “vanilla" Pig system.

5.3 Summary

The work presented in this Chapter, and the contributions made to the area of data-intensive scalable computing, can be summarized as follows:

- We study the fundamental problem of scheduling data processing jobs in multi-tenant, shared Hadoop cluster. Despite the large amount of work that has been dedicated to scheduling (in general, and in particular for Hadoop and MapReduce) only few attempts reached the level of maturity required for adoption in production-level deployments: the most widely used schedulers for Hadoop are the simple First Come First Served (FIFO) approach, and the Hadoop Fair Scheduler, which requires complex (and manual) configuration and achieves system fairness comparable to that of processor sharing. More elaborate techniques have remained an academic exercise due to the inherent difficulty in translating abstract ideas into a practical scheduler. Our approach to the problem is to notice, first, that there is no reason to focus solely on (job) performance or, orthogonally, to system fairness across jobs. To this end, we revisit the well-known (but hardly used in practice, due to stringent requirements on a-priori knowledge of job characteristics) size-based scheduling approach. Specifically we focus on a scheduling discipline that guarantees at the same time system fairness and near-optimal job sojourn times. Using job sojourn time (which includes processing and waiting times) as a performance metric caters system responsiveness, a quality that is highly desirable in production environments where even short (or experimental) jobs are generally executed.

  In our work we proceed with the complete design of the scheduler architecture, which attempts at solving several problems: *i)* job size estimation, which is a required information for (any) size-based scheduling to work; *ii)* lack of preemption primitives, which we address by leveraging the underlying operating system and its support for context switching; and *iii)* a parallel version of the Fair Sojourn Protocol, which is the original scheduling discipline our work builds upon.

  Extensive experiments aiming at assessing the performance of our scheduler under a variety of realistic workloads indicate that the size-based scheduling discipline we implemented considerably lowers the sojourn time for the majority of jobs we executed in our clusters. Our contribution, labeled the Hadoop Fair Sojourn Protocol (HFSP), has been fully implemented and released under the Apache v.2 software license.

- We focus on high-level languages to abstract the complexity of handling both algorithm design and framework tuning while producing data analysis jobs for Hadoop. Specifically, we focused on OLAP systems aiming at computing aggregate measures over de-normalized data. Our work begin with the realization that many aggregate computations operate on hierarchical data, where for example it is important to obtain partial aggregates at different granularities (e.g. compute aggregates on a hourly, monthly and yearly basis). As such, we first focused on the programming model that underlies most (if not all) high-level languages for Hadoop, that is MapReduce. Our contribution consists in a novel design pattern that allows hierarchical data analysis to be translated into a compact MapReduce job, as opposed as several jobs each responsible of a different granularity. The design pattern we define consists in a simple, yet effective, way to route information among worker machines of an Hadoop cluster, and a “pipelined" approach to multi-level computation: in short, our method simply relies on the fact that for some aggregate measures (those that are algebraic) it is possible to amortize the computational cost done at lower layers of the data hierarchy (e.g. hours) and reuse it at higher layers (e.g. days).

  Our on-going work aims at bringing the new design pattern described above to the Pig execution engine, which is a popular high-level abstraction on top of MapReduce. Essentially, we automate the production of jobs that work according to our design pattern (and that thus benefit from the performance enhancements we demonstrate on the legacy MapReduce programming model) by compiling a high-level program (written in Pig Latin) into a single MapReduce job. Currently, we are defining a benchmarking suite to analyze the performance achieved by our compiled jobs with respect to those generated by the “vanilla" distribution of Pig.

In this Chapter we omit several works that have been done in the domain of large-scale data analysis, for the sake of brevity. It is worth to mention that our effort in the domain of data-intensive scalable computing is not only focused on the internals of parallel systems and their performance, but also on the study of data mining techniques to analyze large datasets. In particular, we focused on location-based, social applications to understand user behavior, based on large datasets of user activity. In doing so, we worked on the role played by a “special few" set of users that can be considered as *influentials*. Our work [94, 95] nicely fit the traditional use case for a parallel approach to data analysis, and it is part of our on-going work to extend the single threaded imperative programming model we used to perform our initial analysis to a parallel and functional programming model.

Chapter 6

Teaching

Contents

Introduction

In this Chapter I briefly outline the teaching activity that I carried out since my appointment as an Assistant Prof. at Eurecom. All courses described below are at the Masters level, and the typical audience includes Eurecom's students, Masters students, Ph.D. students and external students coming from Industrial partners of Eurecom's consortium.

Before delving into the details of each course, it is important to emphasize the relation between my teaching and research activities. In my teaching methodology, there exists a feedback loop between research and teaching: in my work as a researcher, I often have the opportunity to relate the problems I tackle to fundamental topics in computer science, which I then develop and include in my courses both as theoretic concepts and as exercises, homeworks or laboratory sessions. It is also the case that – and this is especially true for the most recent course I have created, related to scalable systems for data-intensive processing – some of the laboratory sessions I prepare raise practical problems that I generalize and consider in my research agenda, which thus represent an important source of new problems to tackle.

Exercises, homeworks and, to a large extent, laboratory sessions have played a crucial role in the development of my activities. From the practical standpoint, it is often the case that my group and I have to design, develop and deploy from scratch complex hardware and software systems to setup each laboratory. The complex nature of the software we use and develop also calls for a thorough software development methodology, which benefit both my teaching and research activity: all software sources, but also the lecture notes and slides, are handled through cloud-based software revision control systems that allow to branch, tag, clone and manage complex systems in an easy manner, which is complemented by a documentation taking the form of a Wiki. In addition, the recent developments of my research and teaching activity cover topics that are likely to be relevant for the industry: this is beneficial both to my students, that work on topics likely to offer challenging job opportunities, and to my group. Ph.D. students have the luxury of being able to study relevant problems in the domain of data-intensive computing, and see the impact of their contributions readily adopted by the industry.

To conclude this introductory part to the Chapter, it is important to state what is the evaluation methodology that I use to grade students attending my courses. Due to the applied nature of the majority of the classes I teach, the final grade a student obtains assembles both the understanding of theoretical principles and their application to solve real problems (homework assignments, and laboratory sessions). Ultimately, my goal is to diversify the way I teach to compensate the skew in the distribution of background knowledge my students have: albeit this is a challenging and largely discussed problem, my approach is to level-out the initial heterogeneity of students, and make sure that everyone benefit from my courses.

6.1 Web Technologies

- Role: Professor
- Course Type: Long, 42 hours
- Period, Years: Spring, 2005–2011

The goal of this course is to cover a number of technologies that make up the World Wide Web, ranging from client-side to server-side technologies. The course covers also current best-practices in the deployment of Web applications, including caching layers, load balancing and traditional relational database management systems. Besides covering the basics, the course also surveys major distributed systems and architectures that support today's Web Services and Applications, including distributed data stores (e.g. NoSQL systems).

Class sessions are complemented by external lecturers, working in international and start-up companies where Web Technologies represent a core business. Such lectures include works done at CISCO, PlayAdz (a local startup company in the domain of on-line advertisement), SliceFactory (a local startup company in the domain of Web-based mash-ups and browser extensions).

This course was mandatory for a number of teaching "Tracks" at Eurecom and attracted a lot of students: each year I had roughly 50 students attending the course, and the global ranking from students was excellent.

In 2011, the course has evolved to cover more specific aspects of data-intensive computing, and scalable systems for data storage, thus moving the focus to the back-end rather than to the front-end of Web Technologies. As such, Web Technologies is no longer available in the curriculum, in favor of the last course discussed in this Chapter. This choice also corresponds to an evolution of my research activities, as explained in the introduction.

6.1.1 Term Projects

The Web Technology course requires students to work for a Term Project (performed during the semester and during the laboratory sessions). The goal of the Term Project is for students to familiarize with the technologies illustrated during the lectures but also requires students to be pro-active and come up with original ideas, stemming from using new technologies that have not necessarily been discussed in class. Example of Term Projects are listed below:

- Crawling and Indexing the Web to build a Web Search engine
- Design of a Web Interface for the deployment of Cloud Services
- Implementation of a distributed framework for the Redis key/value store
- Implementation of a simple Recommender Engine for a on-line Movie Broadcast Web Application
- Implementation of a simple Chrome Extension for price comparators

6.2 Algorithm Design

- Role: Professor
- Course Type: Short, 21 hours
- Period, Years: Spring, 2007–

The goals of this course are: *i)* to survey the foundations of the design and analysis of algorithms; *ii)* to provide a practical understanding of complexity theory and algorithms; *iii)* to provide an in-depth understanding of selected problems at the forefront of research explorations in the design and analysis of algorithms. As a prerequisite for this course, basic knowledge of elementary data structures, sorting, and basic terminology involving graphs (including the concepts of depth-first search and breadth-first search) is required, albeit some of these concepts are reviewed in the course. In details, the topics covered in this course are as follows:

- Algorithm design and the notion of Algorithmic Complexity
- Graph Theory: basics notions and advanced concepts related to Network Analysis

- Greedy Algorithms in depth, overview of Divide and Conquer and Dynamic Programming
- Flow Networks and Max-Flow Problems
- Local Search Algorithm and applications to Facility Location Problems
- Randomized Algorithms

This course was not mandatory at Eurecom, but nevertheless attracted roughly 20 students each year. The global ranking from students was excellent, although the preference for a long version of the course, covering more advanced topics has been largely suggested. This point is addressed in Section 6.5, where I elaborate a plan for the evolution of the contents to cover the design and analysis of algorithms to mine massive amount of data.

6.2.1 Exercises

The lectures are complemented by a series of homeworks which may involve the analysis of algorithms at a fairly mathematical level: as such, students are expected to be comfortable reading and writing proofs. Most of the exercises and examples are drawn from problems related to networking and distributed systems, e.g. peer-to-peer systems. Given the small amount of lectures available for this course, the correction and, in general, feedback on exercises and homeworks was mainly done via email.

6.3 Game Theory

- Role: Professor
- Course Type: Short, 21 hours
- Period, Years: Fall, 20010–2012

The goal of this course is for student to get acquainted to basic concepts in Game Theory, together with its algorithmic aspects. Ideas such as dominance, backward induction, Nash equilibrium, evolutionary stability, commitment, credibility, asymmetric information, adverse selection, and signaling are discussed and applied to games played in class and to examples drawn from economics, politics and computer science in general.

This course (and its contents) was highly inspired by one of the first (very popular) on-line courses distributed by the Yale university, department of Economics. Prof. Benjamin Polak was an inspiration to me especially with respect to getting students involved during class.

This course was not mandatory at Eurecom, but nevertheless attracted roughly 30/40 students each year. Recent developments in terms of the research topics covered by my colleagues at Eurecom, made me arrive at the conclusion that – for the sake of coherency – this introductory course to Game Theory had to be merged into a long course that perfectly fit the Teaching Tracks of my Department. Hence, starting from Fall 2013, the course will be part of the Network Economics course, in which I will not be involved due to my different research objectives.

6.3.1 Exercises

The lectures are complemented by a series of homeworks stemming from research papers in the field of computer networks, where game theory is applied to model, analyze and evaluate the operating point of a variety of systems. Such problems are related to the research activity I pursued in the context of peer-to-peer systems, content replication and wireless ad-hoc networks.

6.4 Large-scale Distributed Systems and Cloud Computing

- Role: Professor
- Course Type: Long, 42 hours
- Period, Years: Spring, 2011–

The goal of this course is to provide a comprehensive view on recent topics and trends in distributed systems and cloud computing. The lectures cover software techniques employed to construct and program reliable, highly-scalable systems. This course also covers architecture design of modern data-centers and virtualization techniques that constitute a central topic of the cloud computing paradigm. The main topics treated in the course cover:

- Theory and practice of MapReduce and Hadoop
- Theory and practice of Hadoop Distributed File System
- Design and analysis of parallel algorithms with MapReduce
- Relational Algebra and MapReduce
- Hadoop Pig, Hadoop Hive, Cascading and Scalding
- Theory and practice of BigTable and HBase
- The CAP Theorem
- Amazon Dynamo and Cassandra
- Paxos and Hadoop ZooKeeper

This course is done in collaboration with Prof. Marko Vukolic, who takes care of some of the theoretical aspects developed in the course.

6.4.1 Laboratory Sessions

The course is complemented by a number of laboratory sessions to get hands-on experience with Hadoop and the design of scalable algorithms with MapReduce. In addition, I have prepared exercises to work with high-level languages, with emphasis on Hadoop Pig and Twitter Scalding. Such exercises cover basic statistical analysis of data, as well as complex analytics applied to use-cases including: social network data, telecommunication traffic data, airline traffic data, a large (more than 100 TB) dataset on Web crawls provided `commoncrawl.com`.

The exercises, laboratory set-up, sample input data, and output results have been submitted as open-source projects on the `github.com` website. Students can simply download each laboratory archive, or can contribute to extend and improve exercises and their solutions. This is done via the `git` version control utility, or through a Web interface available on `github.com`.

The infrastructure to carry out each laboratory is conceived as a *private cloud service*: with the help of my research group, we have built from scratch a cluster of powerful servers that run an open-source hyper-visor to achieve virtualization, which are then orchestrated through the OpenStack cloud management open-source software.

6.5 Perspectives

Recently I started to consolidate my teaching and research activity to revolve around data-intensive scalable computing, both at the algorithmic and system level. As such, the courses that I coordinate should be centered on these subjects: this is already the case for the "Clouds" course, but there is still some work to be done concerning the Algorithm Design course.

My teaching agenda is currently focused in promoting a long version of the Algorithm Design class, spanning 14 lectures. The contents of the current version of the course will be compressed to fit the first 4-5 lectures, providing the necessary background to students to understand advanced concepts in the design of scalable algorithms for mining massive amounts of data. The main reference I would like to use in this context is a book from A. Rajaraman and Prof. J. Ullman from Stanford: "Mining of Massive Datasets". The 9-10 new lectures of the Algorithm Design class should cover the following topics:

- Data mining, and its statistical limits (e.g. Bonferroni's Principle)
- Design of parallel algorithms and fundamentals of Functional Programming
- Algorithms for finding similar items
- Mining data streams
- Frequent Itemsets

- Clustering algorithms

Such lectures will be complemented by laboratory sessions with practical applications to recommender systems, advertisement, and data mining.

Chapter 7

Conclusion and perspectives

In this Chapter, I discuss what are the main conclusions to be drawn from the work presented in this manuscript. First, I will focus on the variety of topics covered throughout the past few years: despite the apparent unrelated nature of application domains discussed in this work, I will make the exercise of trying to extract and motivate the underlying research methodology used to address the various problems we tackled, and reposition the contributions in the context of *large scale data management*. I will then conclude with a Section on my research agenda.

Chapter 3 focused on massive-scale content distribution, with application to mobile, wireless networks.[1] In this domain, scalability problems arise due to the inherent "synchronization" that affect human behavior: when consuming content, and because of social-ties and the abundance of communication channels to "spread the word", users flock to a particular information item roughly at the same time, a phenomenon called *flash-crowd*. Hence, systems face a largely unpredictable spike in resource utilization, which calls for a new, decentralized, approach to content distribution, which I discussed in Chapter3. Chapter 4 dealt with massive-scale content storage, that materializes as an Internet application executed by edge devices. Scalability problems developed in two main directions: data size, and number of users. The design of new storage mechanism favoring redundancy instead of replication (specifically because of data size), and new systems whose capacity and performance scale with the number of concurrent users in the system lied at the heart of the work in Chapter 4. Finally, Chapter 5 covered data-intensive applications: machine-learning, statistical modeling, and in general data mining applied to better understand users and the systems or applications they use, constituted the bulk of the use cases for this last chapter. Scalability problems arose because of the scale of data: in this context, moving data (from disk, in the network, across data-centers) is a heavy weight operation that should be avoided as much as possible. Such problems call for a "divide and conquer" approach to computation and new systems to support it.

In summary, scalability problems were a concern for all the applications discussed in the manuscript, and the main objectives of my research were to *understand and improve the way we move, store and process data*. Chapters 3, 4 and 5 dealt with the intricacies in achieving these goals: the solutions I presented had a strong *algorithmic flavor*, in the sense that they all related to well-known and fundamental problems in computer science, ranging from Location Theory and Game Theory to Flow

[1]My work also covered content distribution in the Internet, which I omit from this manuscript for the sake of brevity.

Networks, Scheduling and Resource allocation. Besides the modeling effort required to formulate and study the problems I tackled in this work, it is important to notice that I used a measurement approach to come up with the main motivations to consider such problems and to understand their consequences. As such, the work presented in this manuscript blended theoretic modeling, measurements on real systems, and experimentation, both using synthetic simulations and real-life prototypes.

The main contributions I presented in this manuscript are the following:

- A new view on **distributed replication algorithms**. The work on content dissemination in mobile, wireless networks established a new way of modeling information spreading that is akin to placement problems (in Location Theory) rather than casting it as an epidemic diffusion process or a general caching problem. Essentially, I proposed to use a variant of facility location problems and designed a distributed algorithm capable of coping with several types of dynamics: network fragility due to user mobility and content popularity. Chapter 3 illustrated the properties and performance of the distributed algorithm that I used to find approximate solutions to the problem, which turned out to be very simple to implement, lightweight, and particularly resilient to network and content dynamics;

- A new approach to **data placement and scheduling problems** in wide-area, decentralized storage systems. Due to the inherent need for large amounts of resources, applications such as peer-to-peer on-line storage and backup require sophisticated mechanisms to incite users' altruism: in this context, I proposed a new model in which I viewed data placement as a variant of the *stable matching problem*.[2] Such model was studied through the lenses of Game Theory and I showed, given a set of constraints on user resources, that it was possible to provide incentives to users to increase the "quantity" and "quality" of such resources dedicated to the system, in return for improved performance, with a simple mechanism.

  In addition, I proposed a new method to study the problem of scheduling data transfers among remote machines, when they exhibit an intermittent on-line behavior. This was one of the main problem to address and solve to make on-line backup and storage applications viable and fast. The main idea was to model node availability and data transfer opportunities as a directed bipartite graph, which was then extended to become a flow network. Then, I showed the correspondence between the original problem (of finding the optimal schedule that minimizes data transfer times) and that of the well-known MaxFlow problem, and came up with a polynomial-time algorithm (that builds upon the Ford-Fulkerson algorithm) to find the optimal schedule.

- Size-based scheduling – albeit promoted with vigor by the theoretical community – reached very few instances of real (single-server) systems, not to men-

[2]In this manuscript we glossed over the details of this work, for the sake of brevity.

tion multi-processor environments. One of the main contribution of this line of work, was to show that **size-based scheduling works in practice**, despite size information, which constitutes the input to any size-based protocols, being generally not available. Precisely, I showed that, in a multi-processor system, the hardest problem is not that of computing an accurate estimate of job sizes, but to decide how to allocate resources while estimating job sizes, such that the system can make progress even when size-based scheduling cannot be applied. Furthermore, I showed that multi-processor systems require special care in designing a scheduling algorithm, especially when jobs are complex entities such as MapReduce programs.

- One last contribution of this manuscript was to recognize the need for a **theoretic approach to the design of algorithms** based on the MapReduce (and related) programming model. As I will discuss next, several other researchers have shown that the design of parallel algorithms requires an effort that goes beyond the programming model itself, which means that algorithmic problems and their solutions are rooted in the very nature of the underlying execution framework used to run them. In particular, for the MapReduce programming model, I have shown that well-known operators that are widely used in traditional database management systems required special care when implemented on top of MapReduce, for otherwise a naive implementation (that is agnostic to the underlying execution engine) would suffer severe performance degradation.

Perspectives

In this Section I discuss the main directions that I will take in the next 3 to 5 years. Before delving into the technical details, I first describe the high-level objectives and the "strategy" required to achieve these goals.

Data intensive scalable computing (DISC) constitutes the main domain in which I will focus my research effort. In particular, my goal is to promote a systems approach to research in which, building on solid theoretical foundations, the ultimate goal is to design and implement system prototypes and launch large scale, repeatable experiments to address real problems. An additional trait of my current and forthcoming activity is the definition of and contribution to open-source projects. This approach serves two purposes: first to gain visibility and obtain feedback from the community that will use the "products" of my research; second, and equally important, to overcome current limitations of the scientific community that work on system research: my peers should be able to delve into the internals of how each idea I present is actually implemented (the devil is in the details) and how to repeat the experiments that validate it and assess its performance.

The discussion above requires some additional details on how it can be implemented in practice: in synthesis, there is *i)* a stringent need for hardware material to build clusters of computers to run experiments; *ii)* the requirement to have a team large enough such that the development effort can be well balanced and results can be achieved in a finite amount of time; *iii)* and a proper interaction with the industry. The first two points require substantial funding, which originate mainly from writing and submitting grant proposals. Interaction with the industry, at least in my experience, can be obtained "for free" if the research results that I produce can be of immediate use and applicable to solve engineering problems. In particular, I found that teaching – which in my case is heavily based on research – played a crucial role in attracting industrials: especially the laboratory sessions that I prepared in the past, which have been published as open-source projects accessible to anyone, attracted a large number of industrials, which used them first, and then contacted me to go beyond the basics.

We are now ready to discuss the research directions I plan to pursue next: in this document I will focus on three, inter-related topics.

Scheduling and resource allocation problems

The ideas discussed in Sect. 5.1 of Chapter 5 nicely fit in a more general domain, which deals with scheduling and resource allocation problems in multi-processor systems. This line of work blends theoretical research that strives to model multi-processor systems in general [49, 65, 72, 96] and MapReduce in particular [50, 83], and the intricacies in the design and implementation of scheduling protocols. In particular, I will focus on the problem of scheduling complex jobs in *shared-nothing multi-processor systems*, where the complexity is due to the job structure. When a parallel program is used to go beyond counting, jobs may be composed of multiple sub-jobs, or they may be iterative, which complicate task and job scheduling, especially in a multi-tenant environment whereby concurrent jobs share the same cluster resources.

High-level languages and Work Sharing

As motivated in Sect. 5.2 of Chapter 5, high-level languages represent the natural evolution for (shared-nothing) parallel processing systems such as MapReduce. In this line of work, an important aspect to address is that of "translating", or compiling, a program expressed with a high-level, abstract language to a low-level executable code [54]. High-level abstractions bring, among others, enormous optimization opportunities that are hard (if not impossible) to detect at low-level. For example, by expressing a program as a directed acyclic graph in which each node is an operator and each arc is a data flow (essentially, a logical plan, using the traditional database systems notation), it is possible to significantly improve its performance when executed, by moving operators such that the amount of data flowing in the plan is minimized. In addition to apply well known techniques from the database systems domain, the delay-tolerant nature of MapReduce jobs allow a new type of optimization, which I label *concurrent work-sharing*: the goal is not only to improve performance, but to drastically improve efficiency. In particular, this line of work is related to that of scheduling and aim at minimizing the amount of redundant work done by a compute cluster when multiple jobs are waiting to be served and end up enqueued at the same time in the system.

Parallel algorithm design, with applications to machine learning

Despite it's ubiquitous presence in many recent works on parallel data processing, MapReduce is not a silver bullet, and there has been much work probing its limitations, both from a theoretical perspective [11, 63] and empirically by exploring classes of algorithms that cannot be efficiently implemented with it [19, 25, 48, 109]. Many of these empirical studies take the following form: they present a class of algorithms for which a naively designed algorithm performs poorly, expose it as a fundamental limitation of the MapReduce programming model, and propose an extension or alternative that addresses the limitation. While it is true that a large class of algorithms are not simply amenable to MapReduce implementations, there exist alternative solutions to the same underlying problems that can be easily implemented in MapReduce.

For example, the work in [74] shows that iterative graph algorithms (e.g., PageRank), gradient descent (e.g., for training logistic regression classifiers), and expectation maximization (e.g., for training hidden Markov models, k-means) can all be adapted to the MapReduce programming model. Other works [11] focus on theoretical limits of the programming model and discuss how to design a MapReduce algorithm while taking into account the trade-off that exists between parallelism and communication cost, which are two conflicting aspects in the execution framework that runs MapReduce programs. Yet other works [70, 98] suggest alternative approaches to the design of fundamental graph algorithms (e.g., counting triangles in a graph, minimum spanning trees, etc...) that would otherwise lead to poor performance when implemented in MapReduce. Finally, the work in [63] is the first to propose a computational model for MapReduce problems.

As stated in recent works [74], *"when all you have is a hammer, throw everything that is not a nail"*: I am convinced that we are at the early stages of a proper understanding of the MapReduce programming model, in terms of fundamental research. As such, I will dedicate a considerable amount of work to the consolidation of such computational models, and the definition of a structured methodology toward the design of parallel algorithms for shared-nothing multi-processor systems. In similar terms to what we teach to students in courses such as "Algorithm Design", there should be a solid foundation to the design and analysis of parallel programs written using the functional-style underlying MapReduce. To this end, I will take some noteworthy examples stemming from the machine learning literature, with applications to time-series data analysis, and contribute to the theory alleged above.

Bibliography

[1] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/. 73

[2] Apache hive. http://hive.apache.org/. 77

[3] Cascading. http://www.cascading.org/. 77

[4] Customers Angered as iPhones Overload ATT. http://www.nytimes.com/2009/09/03/technology/companies/03att.html?_r=1. 17

[5] Dropbox. http://dropbox.com/. 57, 58

[6] Hadoop: Open source implementation of MapReduce. http://hadoop.apache.org/. 69

[7] iPhone users eating up ATT's network. http://venturebeat.com/2009/05/11/iphone-users-eating-up-atts-network/. 17

[8] Lib Second Life. www.libsecondlife.org/. 33

[9] Second Life. http://www.secondlife.com. 32

[10] The stratosphere project. https://stratosphere.eu/. 77

[11] F. N. Afrati, A. Das Sarma, S. Salihoglu, and J. D. Ullman. Upper and Lower Bounds on the Cost of a Map-Reduce Computation. *ArXiv e-prints*, June 2012. 101

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley, 2009. 57

[13] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *ACM STOC*, Heraklion, Crete, Greece, July 2001. 22, 23

[14] Arian Baer, Antonio Barbuzzi, Pietro Michiardi, and Fabio Ricciato. Two parallel approaches to network data analysis. In *Proc. of ACM LADIS*, 2011. 78

[15] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/PACTs: A programming model and execution framework for web-scale analytical processing. In *Proc. of ACM SoCC*, 2010. 77

[16] I. Benjamini, G. Kozma, and N. Wormald. The mixing time of the giant component of a random graph, Oct. 2006. 24

[17] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding availability. In *Proc. of USENIX IPTPS*, 2003. 39

[18] Ranjita Bhagwan, Kiran Tati, Yu chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *USENIX NSDI*, 2004. 39, 40

[19] Pramod Bhatotia, Alexander Wieder, İstemi Ekin Akkuş, Rodrigo Rodrigues, and Umut A. Acar. Large-scale incremental data processing with change propagation. In *Proc. of USENIX HotCloud*, 2011. 101

[20] E. W. Biersack, P. Rodriguez, and P. Felber. Performance analysis of peer-to-peer networks for file distribution. In *Proc. Fifth International Workshop on Quality of Future Internet Services (QofIS'04)*, Barcelona, Spain, September 2004. 12

[21] M.C.O. Bogino, P. Cataldi, M. Grangetto, E. Magli, and G. Olmo. Sliding-window digital fountain codes for streaming of multimedia contents. In *IEEE ISCAS*, 2007. 55

[22] J.C. Bolot, T. Turletti, and I. Wakeman. Scalable Feedback Control for Multicast Video Distribution in the Internet. *ACM SIGCOMM Computer Communication Review*, 24(4):58–67, October 1994. 12

[23] Vinayak Borkar, Michael Carey, Raman Grover, Nicola Onose, and Rares Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, Proc. of IEEE ICDE '11, 2011. 77

[24] J.-Y. L. Boudec and M. Vojnovic. Perfect simulation and stationarity of a class of mobility models. In *IEEE INFOCOM*, 2005. 31

[25] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. Haloop: efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3(1-2):285–296, September 2010. 101

[26] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communication & Mobile Computing (WCMC): Special Issue on Mobile Ad Hoc Networking Research, Trends and Applications*, 2002. 31

[27] C. Casetti, C.-F. Chiasserini, M. Fiore, C.-A. La, and P. Michiardi. P2P cache-and-forward mechanisms for mobile ad hoc networks. In *IEEE ISCC*, Sousse, Tunisia, July 2009. 23

[28] P. Cataldi, M. Grangetto, T. Tillo, E. Magli, and G. Olmo. Sliding-window raptor codes for efficient scalable wireless video broadcasting with unequal loss protection. *IEEE TIP*, 19(6), 2010. 55

[29] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, J. Scott, and R. Gass. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Transactions on Mobile Computing*, 2007. 31

[30] A. Chaintreau, A. Mtibaa, L. Massoulie, and C. Diot. The diameter of opportunistic mobile networks. In *ACM CONEXT*, 2007. 31, 32, 35

[31] H. Chang et al. Scheduling in MapReduce-like Systems for Fast Completion Time. In *Proc. of IEEE INFOCOM*, 2011. 69

[32] Biswapesh Chattopadhyay, Liang Lin, Weiran Liu, Sagar Mittal, Prathyusha Aragonda, Vera Lychagina, Younghee Kwon, and Michael Wong. Tenzing a sql implementation on the mapreduce framework. In *Proc. of VLDB*, 2011. 77

[33] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Rec.*, 26(1):65–74, March 1997. 82

[34] H. Chen, Y. Xiao, and X. Shen. Update-based cache access and replacement in wireless data access. *IEEE Trans. on Mob. Comp.*, pages 1734 — 1748, 2006. 19

[35] Y. Chen. We don't know enough to make a big data benchmark suite - an academia-industry view. In *Proc. of WBDB*, 2012. 74

[36] Y. Chen, S. Alspaugh, and R. Katz. Interactive query processing in big data systems: A cross-industry study of mapreduce workloads. In *Proc. of VLDB*, 2012. 68, 74

[37] Y. Chen, A. Ganapathi, R.Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In *Proc. of IEEE MASCOTS*, 2011. 74

[38] Kenjiro Cho et al. Traffic data repository at the wide project. In *Proc. of USENIX ATC*, 2000. 78

[39] B.G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C.H. Papadimitriou, and J. Kubiatowicz. Selfish caching in distributed systems: a game-theoretic analysis. In *ACM PODC*, 2004. 28

[40] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM REVIEWS*, 2007. 34

[41] Bram Cohen. Incentives build robustness in bittorrent. In *Proc. First Workshop on Economics of Peer-to-Peer Systems*, Berkeley, USA, June 2003. 19

[42] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. In *ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002. 26, 27

[43] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proc. of USENIX OSDI*, 2004. 68, 76

[44] A. Derhab and N. Badache. Data replication protocols for mobile ad-hoc networks: A survey and taxonomy. *IEEE Comm. Surveys & Tutorials*, 11(2):33–51, June 2009. 19

[45] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. In *IEEE INFOCOM*, 2007. 40

[46] A. Duminuco, E. Biersack, and T. En-Najjary. Proactive replication in distributed storage systems using machine availability estimation. In *ACM CoNEXT*, 2007. 50

[47] Alessandro Duminuco and Ernst Biersack. Hierarchical codes: How to make erasure codes attractive for peer-to-peer storage systems. In *IEEE P2P*, 2008. 40

[48] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proc. of ACM HPDC*, 2010. 101

[49] Anja Feldmann, JiriÂ Sgall, and Shang-Hua Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130(1):49 – 72, 1994. 100

[50] K. Fox and B. Moseley. Online scheduling on identical machines using srpt. In *In Proc. of ACM-SIAM SODA*, 2011. 100

[51] E.J. Friedman and S.G. Henderson. Fairness and efficiency in web server protocols. In *Proc. of ACM SIGMETRICS*, 2003. 70

[52] Roy Friedman, Gabriel Kliot, and Chen Avin. Probabilistic quorum systems in wireless ad hoc networks. *ACM Trans. on Comp. Syst.*, 28(3), Sept. 2010. 19

[53] A. F. Gates et al. Building a high-level dataflow system on top of map-reduce: The pig experience. In *VLDB*, 2009. 83

[54] Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. Building a high-level dataflow system on top of map-reduce: the pig experience. *Proc. VLDB Endow.*, 2(2):1414–1425, August 2009. 100

[55] A. Ghodsi et al. Dominant resource fairness: Fair allocation of multiple resources types. In *Proc. of USENIX NSDI*, 2011. 69

[56] A V Goldberg and R E Tarjan. A new approach to the maximum flow problem. In *ACM STOC*, 1986. 46

[57] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 2000. 29

[58] T. Hara, Y.-H. Loh, and S. Nishio. Data replication methods based on the stability of radio links in ad hoc networks. In *Database and Expert Systems Applications (DEXA)*, 2003. 19

[59] H. Hayashi, T. Hara, and S. Nishio. On updated data dissemination exploiting an epidemic model in ad hoc networks. In *Lecture Notes in Computer Science*, volume 3853, pages 306–321, Dec. 2006. 19

[60] R. Hekmat and P. Van Mieghem. Degree distribution and hop count in wireless ad hoc networks. In *IEEE International Conference on Networks (ICON)*, pages 603–609, New York, NY, 2003. 24

[61] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, 1996. 31

[62] T. Karagiannis, J.-Y. L. Boudec, and M. Vojnovic. Power law and exponential decay of inter contact times between mobile devices. In *ACM MOBICOM*, 2007. 31

[63] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proc. of ACM SODA*, 2010. 101

[64] Kamal Kc and K. Anyanwu. Scheduling Hadoop jobs to meet deadlines. In *Proc. of CloudCom*, 2010. 69

[65] Svetlana A. Kravchenko and Frank Werner. On a parallel machine scheduling problem with equal processing times. *Discrete Applied Mathematics*, 157(4):848 – 852, 2009. 100

[66] J. Krumm and E. Horvitz. The microsoft multiperson location survey. Technical report, MSR-TR-2005-13, 2005. 32, 35

[67] R. Kumar and K.W. Ross. Peer assisted file distribution: The minimum distribution time. In *Proc. of IEEE HOTWEB*, 2006. 12

[68] C.-A. La, P. Michiardi, C. Casetti, C.-F. Chiasserini, and M. Fiore. Content replication and placement in mobile networks. Technical report, Eurecom Institut, 2011. 24, 27

[69] Chi Anh La and Pietro Michiardi. Characterizing user mobility in Second Life. In *Proc. of ACM SIGCOMM WOSN*, 2008. 35

[70] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *Proc. of ACM SPAA*, 2011. 101

[71] S. Le Blond, F. Le Fessant, and E. Le Merrer. Finding good partners in availability-aware p2p networks. *Stabilization, Safety, and Security of Distributed Systems*, pages 472–484, 2009. 41

[72] Jan Karel Lenstra, David B. Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990. 100

[73] Keqin Li. Performance analysis and evaluation of random walk algorithms on wireless networks. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPSW)*, pages 1–8, Atlanta, GA, Apr. 2010. 24

[74] J. Lin. MapReduce is Good Enough? If All You Have is a Hammer, Throw Away Everything That's Not a Nail! *ArXiv e-prints*, September 2012. 101

[75] L. Lovasz. Random walks on graphs: A survey. *Combinatorics*, 2:1–46, 1993. 24

[76] M. Luby. LT codes. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 271–280. IEEE, 2002. 55, 56

[77] J. Luo, P.T. Eugster, and J.-P. Hubaux. PILOT: Probabilistic lightweight group communication system for mobile ad hoc networks. *IEEE Trans. on Mob. Comp.*, 3(2), Apr.-June 2004. 19

[78] Thomas Mager, Ernst Biersack, and Pietro Michiardi. A measurement study of the Wuala on-line storage service. In *Proc. of IEEE P2P*, 2012. 38

[79] Marco Mellia, Andrea Carpani, and Renato Lo Cigno. Tstat: Tcp statistic and analisys tool. In *IEEE QoSIP*, 2003. 78

[80] P. Michiardi, C.-F. Chiasserini, C. Casetti, C.-A. La, and M. Fiore. On a selfish caching game. In *ACM PODC (Brief Announcement)*, Calgary, Canada, Aug. 2009. 30

[81] P. Michiardi and G. Urvoy-Keller. Performance analysis of cooperative content distribution for wireless ad hoc networks. In *IEEE WONS*, 2007. 19

[82] T. Moscibroda and R. Wattenhofer. Facility location: Distributed approximation. In *ACM PODC*, Las Vegas, NV, July 2005. 22

[83] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlos. On scheduling in map-reduce and flow-shops. In *In Proc. of ACM SPAA*, 2011. 100

[84] W. Navid and T. Camp. Stationary distributions for the random waypoint model. *IEEE Transactions on Mobile Computing*, 2004. 31

[85] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *ACM SIGMOD*, 2008. 77, 83

[86] Lluis Pamies-Juarez and Pedro Garcia-Lopez. Maintaining data reliability without availability in p2p storage systems. In *ACM SAC*, 2010. 50

[87] Mario Pastorelli, Antonio Barbuzzi, Damiano Carra, Matteo Dell'Amico, and Pietro Michiardi. Practical Size-based Scheduling for MapReduce Workloads. Technical report, 2013. `http://arxiv.org/abs/1302.2749`. 70, 71, 73

[88] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proc. of ACM SIGCOMM*, 2004. 12

[89] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM'01*, San Diego, California, USA, August 27-31 2001. 12

[90] K. Ren, Y. Kwon, M. Balazinska, and B. Howe. Hadoop's adolescence: A comparative workload analysis from three research clusters. In *Technical Report, CMU-PDL-12-106*, 2012. 68

[91] I. Rhee, M. Shin, S. Hong, K. Lee, and S. Chong. On the levy-walk nature of human mobility. In *IEEE INFOCOM*, 2008. 32, 35

[92] T. Sandholm and K. Lai. Dynamic proportional share scheduling in Hadoop. In *Proc. of JSSPP*, 2010. 69

[93] K. Sbai, C. Barakat, J. Choi, A. Al Hamra, and T. Turletti. Adapting BitTorrent to wireless ad hoc networks'. *AdHoc Now*, 2008. 19

[94] Xiaolan Sha, Daniele Quercia, Matteo Dell'Amico, and Pietro Michiardi. Trend makers and trend spotters in a mobile application. In *Proc. of IEEE CSCW*, 2013. 88

[95] Xiaolan Sha, Daniele Quercia, Pietro Michiardi, and Matteo Dell'Amico. Spotting trends: the wisdom of the few. In *ACM RecSys*, 2012. 88

[96] D. Shmoys, J. Wein, and D. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, 1995. 100

[97] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM'01*, San Diego, California, USA, August 27-31 2001. 12

[98] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proc. of ACM WWW*, 2011. 101

[99] Laszlo Toka. *Game theoretic analysis of distributed systems : design and incentives*. PhD thesis, Thesis, 01 2011. 43, 48, 56

[100] Laszlo Toka and Pietro Michiardi. Analysis of user-driven peer selection in peer-to-peer backup and storage systems. In *ACM VALUETOOLS*, 2008. 38

[101] Laszlo Toka and Pietro Michiardi. A dynamic exchange game. In *ACM PODC*, 2008. 38

[102] Laszlo Toka and Pietro Michiardi. Analysis of user-driven peer selection in peer-to-peer backup and storage systems. *Telecommunication Systems*, 2011. 38

[103] Daniel Warneke and Odej Kao. Nephele: Efficient parallel data processing in the cloud. In *MTAGS*, 2009. 77

[104] Robin Wauters. Online backup company carbonite loses customers' data, blames and sues suppliers. TechCrunch, 2009. 57

[105] F. Lam W.L. Tan and W.C. Lau. An empirical study on the capacity and performance of 3g networks. *IEEE Transactions on Mobile Computing*, 2007. 29

[106] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin. FLEX: A slot allocation scheduling optimizer for MapReduce workloads. In *Proc. of ACM MIDDLEWARE*, 2010. 69

[107] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *IEEE INFOCOM*, 2003. 31

[108] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proc. of ACM EuroSys*, 2010. 69, 71, 72, 73, 74

[109] Yanfeng Zhang, Qixin Gao, Lixin Gao, and Cuirong Wang. Priter: a distributed framework for prioritized iterative computations. In *Proc. of ACM SOCC*, 2011. 101