Distributed Optimization and Data Market Design

Thesis by
Palma London

In Partial Fulfillment of the Requirements for the
degree of
Master of Science

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2017
Submitted May 19, 2017

ACKNOWLEDGEMENTS

I thank Prof. Adam Wierman for advising me throughout my Masters Degree. I also thank Shai Vardi, Xiaoqi Ren, Niangjun Chen, and Juba Ziani for their help during collaborations.

ABSTRACT

In this thesis we propose a new approach for distributed optimization based on an emerging area of theoretical computer science – local computation algorithms. The approach is fundamentally different from existing methodologies and provides a number of benefits, such as robustness to link failure and adaptivity to dynamic settings. Specifically, we develop an algorithm, LOCO, that given a convex optimization problem P with n variables and a "sparse" linear constraint matrix with m constraints, provably finds a solution as good as that of the best online algorithm for P using only $O(\log(n+m))$ messages with high probability. The approach is not iterative and communication is restricted to a localized neighborhood. In addition to analytic results, we show numerically that the performance improvements over classical approaches for distributed optimization are significant, e.g., it uses orders of magnitude less communication than ADMM.

We also consider the operations of a geographically distributed cloud data market. We consider design decisions that include which data to purchase (data purchasing) and where to place or replicate the data for delivery (data placement). We show that a joint approach to data purchasing and data placement within a cloud data market improves operating costs. This problem can be viewed as a facility location problem, and is thus NP-hard. However, we give a provably optimal algorithm for the case of a data market consisting of a single data center, and then generalize the result from the single data center setting in order to develop a near-optimal, polynomial-time algorithm for a geo-distributed data market. The resulting design, Datum, decomposes the joint purchasing and placement problem into two subproblems, one for data purchasing and one for data placement, using a transformation of the underlying bandwidth costs. We show, via a case study, that Datum is near-optimal in practical settings.

PUBLISHED CONTENT AND CONTRIBUTIONS

Palma London, Niangjun Chen, Shai Vardi, and Adam Wierman. *Distributed Optimization via Local Computation Algorithms.* `http://users.cms.caltech.edu/~plondon/loco.pdf`. Under submission. 2017.

P. London and S. Vardi came up with the results and proofs in this paper, and P. London coded and ran all experiments. Article adapted and extended for this thesis.

Xiaoqi Ren, Palma London, Juba Ziani, and Adam Wierman. *Joint Data Purchasing and Data Placement in a Geo-Distributed Data Market.* Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science. 2016.

X. Ren and P. London came up with the results and proofs in this paper, and P. London coded and ran all experiments. Article adapted and extended for this thesis.

TABLE OF CONTENTS

Part I

Introduction and Motivation

Chapter 1

INTRODUCTION

We consider algorithms for distributed optimization and their applications. In this thesis we propose two new approaches to distributed optimization and consider an exciting application in distributed data markets.

The first algorithm, LOCO, is a fundamentally new approach to distributed optimization. There are a wide variety of approaches for distributed optimization, which fall into the categories of dual decomposition and subgradient methods, and consensus-based schemes. We propose a new approach which utilizes local computation algorithms, a rising field in theoretical computer science. A local algorithm is one where a query about part of a solution to a problem can be answered by communicating with only a small number of computation units in the distributed setting. Neither iterative descent methods nor consensus methods are local: answering a query about a part of the solution requires global communication. The advantage offered by LOCO is that significantly less communication is required to solve the optimization problem [60].

Secondly, we consider the management of geographically distributed data center. For example, imagine we are operating a data service like Yelp. Clients submit queries to Yelp on their personal devices. Then, Yelp contacts various data providers that sell data and may have to pay a data purchasing fee. Yelp routes data through its data centers, and delivers it to clients. We believe that data as a service is a growing market. In the near future people might want to buy data as a service just as computing infrastructure is bought today. Given prices offered by data providers, we, the data center designers, need to decide which data providers to buy data from to satisfy client queries at minimal cost. We also decide how data should be stored and replicated throughout the geo-distributed data center to minimize bandwidth and latency costs. We design a data center that jointly optimizes data purchasing costs and bandwidth costs. Today many data center designs minimize bandwidth costs. However, when data purchasing costs are also considered, the structure of the optimization problem changes. We model the problem as a facility location problem and

thus it is NP hard. We propose a near-optimal, polynomial-time algorithm and in a simulation study we show that our algorithm is near optimal [77].

Part II

Distributed Optimization via Local Computation Algorithms

Chapter 2

INTRODUCTION TO DISTRIBUTED OPTIMIZATION

The goal of this work is to introduce a new, fundamentally different approach to distributed optimization based on an emerging area of theoretical computer science – local computation algorithms.

Distributed optimization is an area of crucial importance to networked control. Settings where multiple, distributed, cooperative agents need to solve an optimization problem to control a networked system are numerous and varied. Examples include management of content distribution networks and data centers [13, 70], communication network protocol design [47, 62, 84], trajectory optimization [39, 53], formation control of vehicles [86, 75], sensor networks [69, 59], control of power systems [27, 72], and management of electric vehicles and distributed storage devices [16, 35].

Distributed optimization is a field with a long history. Beginning in the 1960s approaches emerged for solving large scale linear programs via decomposition into pieces that could be solved in a distributed manner. For example, two early approaches are Bender's decomposition [9] and the Dantzig-Wolfe decomposition [24, 23], which can both be generalized to nonlinear objectives via the subgradient method [10, 67, 83].

Today, there is a wide variety of approaches for distributed optimization, e.g., primal decomposition [54, 10] and dual decomposition [28, 67, 61, 84]. See [71] for a survey. Broadly, these approaches tend to fall into two categories. The first category uses dual decomposition and subgradient methods [47, 61, 84]; the second involves consensus-based schemes which enable decentralized information aggregation, which forms the basis for many first order and second order distributed optimization algorithms [12, 66].

While the algorithms described above are *distributed*, they are not *local*. A local algorithm is one where a query about a small part of a solution to a problem can be answered by communicating with only a small neighborhood around the part queried[1] (see Subsection 2.2 for a more comprehensive defini-

[1] 'Local' is an overloaded term in the literature. We mean local in the sense of [78].

tion and example). Clearly, neither iterative descent methods nor consensus methods are local: answering a query about a piece of the solution requires global communication.

Local computation is well suited for distributed optimization. For example, any failure in the system only has local effects: if a node in a distributed system goes offline while an iterative distributed algorithm is executing, the whole process is brought to a halt (or at least the system needs to be carefully designed to be able to accommodate such failures); if the computations are all local, the failure will only affect a small number of nodes in the neighborhood of the failure. Similarly, lag in a single edge affects the computation of the entire solution in the iterative setting, while most computations are not be affected at all when the computations are local. Another advantage of local computation is that it allows the system to be more dynamic: an arrival of another node requires recomputing the entire solution if the algorithm is not local, but requires only a few local messages and computations if the algorithm is local.

Despite the benefits of local algorithms for distributed optimization, the problem of designing a local, distributed optimization algorithm is open.

2.1 Contributions of this work

This paper introduces an algorithm, LOCO, (LOcal Convex Optimization) that is both *distributed* and *local*. It is not an iterative method and uses far less communication to compute small parts of the solution than iterative descent and consensus methods, e.g., ADMM and dual decomposition, while matching the total communication if the whole solution is queried.

While the technique we propose is general, in this work, we focus on a canonical optimization problem: network utility maximization. Due to space restrictions, we only consider the variant of maximizing throughput, which amounts to solving a distributed linear program. We focus on this case because it is particularly well-studied and, in addition, the objective function is linear, which in many cases is known to produce the worst performance guarantee for online convex optimization problems [5, 41].

In Section 4, we provide worst-case guarantees on the performance of LOCO with respect to the relative error and the number of messages it requires. In Section 5, we compare the performance of LOCO with ADMM, and show that

LOCO uses orders of magnitude less communication than ADMM if only part of the solution is required, and the same order of magnitude if the entire solution is required. Furthermore, in terms of both the amount of communication required and the relative error, LOCO vastly outperforms its theoretical guarantees.

The key idea behind LOCO is an extension of recent results from the emerging field of *local computation algorithms* (LCA) in theoretical computer science (e.g., [63, 76, 56]). In particular, a key insight of the field is that online algorithms can be converted into local algorithms in graph problems with bounded degree [63]. However, much of the focus of local algorithms has, to this point, been on graph problems (see related literature below). The technical contribution of this work is the extension of these ideas to convex programs.

2.2 Related literature

This work, for the first time, brings techniques from the field of local computation algorithms into the domain of networked control. The LCA model was formally introduced by Rubinfeld et al. [78], after many algorithms fitting within the framework had recently appeared in distinct areas, e.g., [79, 4, 46]. LCAs have received increasing attention in the years that followed as the importance of local, distributed computing has grown with the increasing scale of problems in distributed systems, the internet of things, etc.

The main idea of LCAs is to compute a piece of the solution to some algorithmic problem using only information that is close to that piece of the problem, as opposed to a global solution, by exchanging information across distributed agents. More concretely, an LCA receives a *query* and is expected to output the part of the solution associated with that query. For example, an LCA for maximal matching would receive as a query an edge, and its output would be "yes/no", corresponding to whether or not the edge is part of the required matching. The two requirements are (i) the replies to all queries are consistent with the same solution, and (ii) the reply to each query is "efficient", for some natural notion of efficient.

Most of the work on LCAs has focused on graph problems such as matching, maximal independent set, and coloring (e.g., [3, 56, 76, 31]) and the efficiency criteria were the number of probes to the graph, the running time and the

amount of memory required. This paper extends the LCA literature by moving from graph problems to optimization problems, which have not been studied in the LCA community previously. Mansour et al. [63] showed a general reduction from LCAs to online algorithms on graphs with bounded degree. The key technical contribution of our the work is extending that technique to design LCAs for convex programs. In contrast to previous work whose primary focus was probe, time and space complexities, the efficiency criterion we use is the number of messages required as this is usually the expensive resource in networked control.

Chapter 3

NETWORK UTILITY MAXIMIZATION

In order to illustrate the application of local computation algorithms to distributed optimization, we focus on the classic setting of network utility maximization (NUM). The NUM framework is a general class of optimization problems that has seen wide-spread application to distributed control in domains from the design of TCP congestion control [47, 61, 62, 84] to understanding of protocol layering as optimization decomposition [18, 71] and power system demand response [80, 58]. For a recent survey, see [96].

3.1 Model

The NUM framework considers a network containing a set of links $\mathcal{L} = \{1, \ldots, m\}$ of capacity c_j, for $j \in \mathcal{L}$. A set of $\mathcal{N} = \{1, \ldots, n\}$ sources shares the network; source $i \in \mathcal{N}$ is characterized by $(L_i, f_i, \underline{x}_i, \bar{x}_i)$: a path $L_i \subseteq \mathcal{L}$ in the network; a (usually) concave utility function $f_i : \mathbb{R}_+ \to \mathbb{R}$; and the minimum and maximum transmission rates of i.

The goal in NUM is to maximize the sources' aggregate utility. Source i attains a concave utility $f_i(x_i)$ when it transmits at rate x_i that satisfies $\underline{x}_i \leq x_i \leq \bar{x}_i$; the optimization of aggregate utility can be formulated as follows,

$$\begin{aligned} \max_{x} \quad & \sum_{i=1}^{n} f_i(x_i) \\ \text{subject to} \quad & Ax \leq c \\ & \underline{x} \leq x \leq \bar{x}, \end{aligned}$$

where $A \in \mathbb{R}_+^{m \times n}$ is defined as $A_{ji} = \begin{cases} 1, & j \in L(i) \\ 0, & \text{otherwise} \end{cases}$.

The NUM framework is general in that the choice of f_i allows for the representation of different goals of the network operator. For example, using $f_i(x_i) = x_i$, maximizes throughput; setting $f_i(x_i) = \log(x_i)$ achieves proportional fairness among the sources; setting $f_i(x_i) = -1/x_i$ minimizes potential delay; these are common goals in communication network applications [61, 64].

In this paper we focus on the throughput maximization case, i.e., $f_i(x_i) = x_i$; in this case NUM is an LP. Note that the classical dual decomposition approach does not work for throughput maximization since it requires the objective function to be strictly concave. However, ADMM can be applied.

Our complexity results hinge on the assumption that the constraint matrix A is sparse. The *sparsity* of A is defined as $\max\{\alpha, \beta\}$, where α and β denote the maximum number of non-zero entries in a row and column of A respectively. Formally, we say that A is *sparse* if the sparsity of A is bounded by a constant. This assumption usually holds in network control applications since α is the maximum number of sources sharing a link, which is typically small compared to n, and β is the maximum number of links each source uses, which is typically small compared to m.[1]

3.2 Distributed Algorithms for Network Utility Maximization

Given the NUM formulation above, the algorithmic goal is to design a protocol that efficiently finds an (approximately) optimal solution. If the network is huge, it is often beneficial to distribute the solution, as performing the entire computation on a single machine is too costly [14, 84].

There is a large literature across the networked control and communication networks literatures that seeks to design such distributed optimization algorithms, e.g., [18, 47, 62]. Dual decomposition algorithms are particularly prominent for use in this setting. However, many such methods cannot be applied to the case of throughput maximization, i.e., linear f_i. One extremely prominent algorithm that does apply in the case of throughput maximization is Alternating Method of Multipliers (ADMM), which was introduced by [34] and has found broad applications in e.g., denoising images [85], support vector machine [33], and signal processing [20, 19, 81]. As a result, we use ADMM as a benchmark for comparison in this paper. For completeness, the application of ADMM to NUM is described in Appendix A.1.

3.3 Performance metrics

Distributed algorithms for NUM should perform well on two measures.

The first is *message complexity*: the number of messages that are sent across

[1] When α is large, many links will be congested and all sources will experience greater delay, the routing protocol (IP) will start using different links; also, due to the small diameter of the Internet graph [2], β is small compared to m.

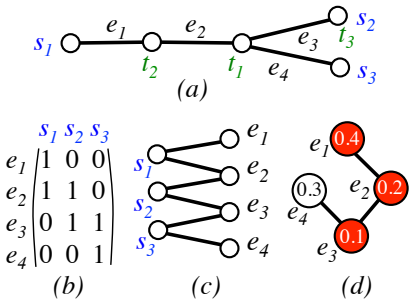


Figure 3.1: An illustration of LOCO on a toy graph with five nodes and four edges, $e_1, \ldots, e_4$. There are three sources, s_1, s_2, s_3, with paths ending in destinations t_1, t_2, t_3 respectively. The graph is depicted in (a); the constraint matrix for NUM is given in (b); the bipartite graph representation of the matrix in (c); and the dependency graph in (d). The rank of each constraint (edge) is written in the node representing the constraint in the dependency graph. The shaded nodes represent the query set for source s_1.

links of the network in order to compute the solution. When the algorithm uses randomization, we want the message complexity to hold with probability at least $1 - \frac{1}{n^\alpha}$, where where n is the number of vertices in the network and $\alpha > 0$ can be an arbitrarily large constant. We denote this by $1 - \frac{1}{\text{poly}\, n}$. We do not bound the size of the messages, but note that in both our algorithm and ADMM the message length will be of order $O(\log n)$.

The second is the *approximation ratio*, which measures the quality of the solution provided by the algorithm. Specifically, an algorithm is said to α-approximate a maximization problem if its solution is guaranteed to be at least $\frac{OPT}{\alpha}$, where OPT is the value of the optimal solution. If the algorithm is randomized, the approximation ratio is with respect to the expected size of the solution. We will compare the performance of LOCO with iterative algorithms such as ADMM, for which approximation ratio is not a standard measure. Thus in our empirical results, comparison with the optimal solution is made using *relative error*, defined in Section 5.1, which is related to, but slightly different from the approximation ratio.

Chapter 4

LOCAL CONVEX OPTIMIZATION

In this section, we introduce a local algorithm for distributed convex optimization, *LOcal Convex Optimization* (LOCO). In LOCO, every source in the network computes its portion of a near optimal solution using a small number of messages, without needing global communication or iteration. This is in contrast to iterative descent methods, e.g. ADMM, which are *global*, i.e., they spread the information necessary to find an optimal solution throughout the whole network over a series of rounds. LOCO has provable worst-case guarantees on both its approximation ratio and message complexity, and improves on the communication overhead of iterative descent methods by orders of magnitude in practice when asked to compute a piece of the optimal solution.

4.1 An overview of LOCO

The key insight in the design and analysis of LOCO is that any natural[1] online optimization algorithm can be converted into a local, distributed optimization algorithm. Note that the resulting distributed algorithm is for a static problem, *not* an online one. Further, after this conversion, the distributed optimization algorithm has the same approximation ratio as the original online optimization algorithm. Thus, given an optimization problem for which there exist effective online algorithms, these online algorithms can be converted into effective local, distributed algorithms.

More formally, to reduce a static optimization problem to an online optimization problem, we do the following. Let Y be the set of constraints of an optimization problem P. Let $r : Y \to [0, 1]$ be a ranking function that assigns each constraint y_j a real number between 0 and 1, uniformly at random. We call $r(y_j)$ y_j's *rank*. Suppose that there is some online algorithm ALG that receives the constraints sequentially and must augment the variables immediately and

[1]Strictly speaking, we require that the online algorithm have the following characteristic: knowing the output of the algorithm for the "neighbors" of a query q that arrived before q is sufficient to determine the output for q. We omit this technicality from the theorem statements as the online algorithm we use, and indeed all online algorithms for convex optimization that we are aware of, have this property. For a more in-depth discussion, we refer the reader to [76].

irrevocably so as to satisfy each arriving constraint. Suppose furthermore that for each constraint y_j, we can pinpoint a small set of constraints $S(y_j)$ (which we call y_j's *query set*) that arrived before it so that restricting the set of constraints of P to $S(y_j)$ results in ALG producing (exactly) the same solution for the variables that are present in y_j. Then simulating ALG only on $S(y_j)$ would suffice to obtain the solution for the variables in y_j. This is precisely what our algorithm does: it generates a random order of arrival for the constraints, and for each constraint y_j, it constructs such a set $S(y_j)$ and simulates the online algorithm on it. An arbitrary ordering could mean that these dependency sets are very large for some constraints; to bound the size of these sets, we require that (i) the constraint matrix of P is sparse and (ii) the order generated is random.[2]

Concretely, there are two main steps in LOCO. In the first, LOCO generates a localized neighborhood for each vertex. In the second, LOCO simulates an online algorithm on the localized neighborhood. Importantly, the first step is independent of the precise nature of the online algorithm, and the second is independent of the method used to generate the localized neighborhoods. Therefore, we can think of LOCO as a general methodology that can yield a variety of algorithms. For example, we can use different online algorithms for the second step of LOCO depending on whether we consider a linear NUM problem or a strictly convex NUM problem. More specifically, the two steps work as follows.

Step 1, Generating a localized neighborhood For clarity, we break the first step into three sub-steps, see also Figure 3.1.

Step 1a, Representing the constraint matrix as a bipartite graph A boolean matrix A can be represented as a bipartite graph $G = (L, R, E')$ as follows. Each row of A is represented by a vertex $v_\ell \in L$; each column by a vertex $v_r \in R$. The edge (v_ℓ, v_r) is in E' if and only if $A_{\ell,r} = 1$. A more intuitive way to interpret G is the following: L represents the variables, R the constraints. Edges represent which variables appear in which constraints. Note that the maximum degree of G is exactly the sparsity of A.

[2]Pseudo-random orders suffice, see [76].

Step 1b, Constructing the dependency graph We construct the *dependency graph* $H = (V, E)$ as follows. The vertices of the dependency graph are the vertices of R; an edge exists between two vertices in H if the corresponding vertices in G share a neighbor. Intuitively, H represents the "direct dependencies" between the constraints: changing the value of any variable immediately affects all constraints in which it appears, hence these constraints can be thought of as directly dependent. The maximum degree of H is upper bounded by the square of the sparsity of A.

Step 1c, Constructing the query set In order to build the query set, we generate a random *ranking function* on the vertices of H, $r : V \to [0, 1]$. Given the dependency graph H, an initial node $y \in V$ and the ranking function r, we build the *query set* of y, denoted $S(y)$, using a variation of BFS, as follows. Initialize $S(y)$ to contain y. For every vertex $v \in S(y)$, scan all of v's neighbors, denoted $N(v)$. For each $u \in N(v)$, if $r(u) \leq r(v)$, add u to $S(y)$. Continue iteratively until no more vertices can be added to $S(y)$ (that is, for every vertex $v \in S(y)$ all of its neighbors that are not themselves in $S(y)$ have lower rank than v). If there are ties (i.e., two neighbors u, v such that $r(u) = r(v)$), we tie-break by ID.[3]

Step 2, Simulating the online algorithm Assume that we have an online algorithm for the problem that we would like LOCO to solve (in this paper we use the online packing Algorithm of Buchbinder and Naor [15, chapter 14]. We provide the pseudocode in Appendix A, for completeness). The specific setting that the online algorithm must apply to is the following: the variables of the convex program are known in advance, as are the univariate constraints. The (rest of the) constraints arrive one at a time; the online algorithm is expected to satisfy each constraint as it arrives, by increasing the value of some of the variables. It is never allowed to decrease the value of any variable. We simulate the online algorithm as follows:

In order to compute its own value in the solution, source i applies r to the set of constraints in which it is contained, $Y(i)$. For $y = \arg\max_{z \in Y(i)}\{r(z)\}$, it simulates the online algorithm on $S(y)$. That is, it executes the online algorithm on the neighborhood constructed in Step 1 for the "last arriving"

[3] Any consistent tie breaking rule suffices.

constraint that contains i. i's value is the value of i at the end of the simulation. Claim 4 below shows that i's value is identical to its value if the online algorithm was executed on the entire program, with the constraints arriving in the order defined by r.

4.2 Analysis of LOCO

Our main theoretical result shows that LOCO can compute solutions to convex optimization problems that are as good as those of the best online algorithms for the problems using very little communication. We then specialize this case to throughput maximization in NUM. While we focus on NUM in this paper, the theorem (and its proof) apply to a wider family of problems as well. Specifically, the conversion from online to local outlined below can be used more broadly for any class of optimization problems for which effective online algorithms exist. Thus, improvements to online optimization problems immediately yield improved local optimization algorithms.

Theorem 1 *Let P be a problem with a concave objective function and linear inequality constraints, with n variables and m constraints, whose constraint matrix has sparsity σ. Given an online algorithm*[4] *for P with competitive ratio $h(n, m)$, there exists a local computation algorithm for P with approximation ratio $h(n, m)$ that uses $2^{O(\sigma^2)} \log(n + m)$ messages with probability $1 - 1/poly(n, m)$.*

In particular, we have the following result, for NUM with a linear objective function.

Theorem 2 *Let P be a throughput maximization problem with n variables, m constraints, and a sparse constraint matrix. LOCO computes an $O(\log m)$ – approximation to the optimal solution of P using $O(\log(n+m))$ messages with probability $1 - 1/\operatorname{poly}(n, m)$.*

The approximation ratio in Theorem 2 comes from the online algorithm presented and analyzed in [15] (see Lemma 6). The analysis of the online algorithm is for adversarial input; therefore it is natural to expect LOCO to achieve a much better approximation ratio in practice, as LOCO randomizes

[4] See footnote [1].

the order in which the constraints "arrive". It is an open question to give better theoretical bounds for stochastic inputs, and if such results are obtained they would immediately improve the bounds in Theorem 2.

The core technical lemma required for the proof of Theorem 1 is the following.

Lemma 3 *Let $G = (V, E)$ be a graph whose degree is bounded by d and let $r : V \to [0, 1]$ be a function that assigns to each vertex $v \in V$ a number between 0 and 1 independently and uniformly at random. Let T_{max} be the size of the largest query set of G: $T_{max} = \max\{|T_v| : v \in V\}$. Then, for $\lambda = 4(d + 1)$,*

$$\Pr[|T_{max}| > 2^{\lambda} \cdot 15\lambda \log n] \leq \frac{1}{n^2}.$$

The proof of Lemma 3 uses ideas from a proof in [76], and employs a *quantization* of the rank function. Its proof is deferred to Appendix B. The following simple claim implies that the approximation ratio of LOCO is the same as that of the online algorithm.

In addition to Lemma 3, the following claim and technical lemma are needed to complete the proof of Theorem 2.

Claim 4 *For any source i, the value of x_i in the output of LOCO is identical to its value in the output of the online algorithm.*

Proof 5 *Let constraint y_j be the last constraint containing i that arrives in the order defined by r; its arrival is the last time i will be updated. Therefore it is sufficient to only consider constraints arriving before y_j. Further, by design, $S(y_j)$ is the set of constraints at whose arrival there is possibly some change that may affect the value of i.*

The following lemma is a restatement of Theorem 14.1 in [15], adapted to throughput maximization. See Appendix A for the pseudocode of the algorithm.

Lemma 6 *For any $B > 0$, there exists is a B-competitive online algorithm to linearly-constrained NUM with m constraints; each constraint is violated by a factor at most $\frac{2\log(1+m)}{B}$.*

Proof 7 (Proof of Theorem 2) *Theorem 1, Claim 4 and Lemma 6, setting $B = 2\log(1 + m)$, imply Theorem 2.*

4.3 Contrasting LOCO and ADMM

LOCO fundamentally differs from iterative descent and consensus style approaches to distributed optimization. While iterative descent and consensus style approaches are inherently iterative, LOCO is not. Under LOCO, a node can compute its value in one shot, once it gets information about its query set.

Additionally, while iterative descent and consensus style approaches are *global*, LOCO is local. Under LOCO, communication stays within the query set and so the computation only needs to be updated if changes happen within the query set. This means that LOCO is robust to churn, failures, and communication problems outside of that set of nodes.

Another important difference is that LOCO does not compute the optimal solution, while iterative descent and consensus style approaches will eventually converge to the true optimal. The proven analytical bounds for LOCO are based on worst-case *adversarial* input. We show in Section 5.2 that our empirical results outperform the theoretical guarantees by a considerable margin. This is in part because the ranking is done randomly rather than in an adversarial fashion (we elaborate on this in Section 5).

Finally, note that there is an important difference in the form of the theoretical guarantees for LOCO and iterative descent and consensus style algorithms. LOCO has guarantees in terms of the approximation ratio, while iterative descent and consensus style algorithms have convergence rate guarantees. For example, ADMM has guarantees on convergence of the norms of the primal and dual residuals [14, Chapter 3.3].

Chapter 5

CASE STUDY

Here we present the results of a simulation study demonstrating the empirical performance of LOCO on both synthetic and real networks. The results highlight that an orders-of-magnitude reduction in communication is possible with LOCO as compared to ADMM, which we choose as a prominent example of current approaches for distributed optimization. For concreteness, our experiments focus our numeric results on distributed linear programming, i.e., the case of linear NUM. This is the NUM setting where one could expect LOCO to perform the worst, given that linear functions are typically the worst-case examples for online convex optimization algorithms [5, 41].

5.1 Experimental setup

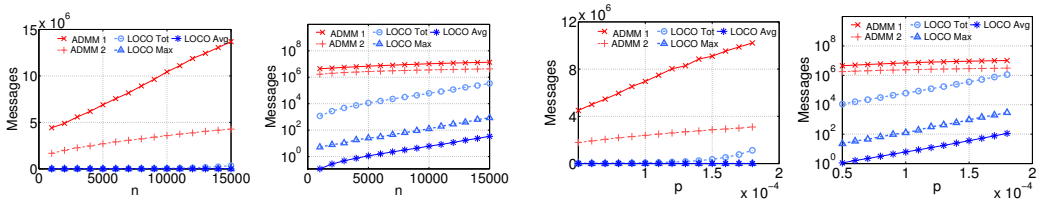
Problem Instances

For our first set of experiments, we generate random synthetic instances of linear NUM. Let $n = m$ and define the constraint matrix $A \in \mathbb{R}^{(m \times n)}$ as follows. Set $\tilde{A}_{j,i} = 1$ with probability p and $\tilde{A}_{j,i} = 0$ otherwise. Let $A = \tilde{A} + I_n$ to ensure each row of A has at least one non zero entry.[1] The vector $c \in \mathbb{R}^n$ is drawn *i.i.d.* from Unif$[0, 1]$. We set the minimum and maximum transmission rates to be $\underline{x}_i = 0$ and $\bar{x}_i = 1$. Finally, for the rank function used by LOCO we use a random permutation of the vertex IDs.[2]

For our second set of experiments, we use the real network from the graph of Autonomous System (AS) relationships in [87]. The graph has 8020 nodes and 36406 edges. In order to interpret the graph in a NUM framework, we associate each source with a path of links, ending at a destination node. To do this, for each node i in the graph, we randomly select a destination node t_i which is at distance ℓ_i, sampled *i.i.d.* from Unif$[\ell - 0.5\ell, \ell + 0.5\ell]$. We repeat this for several values of ℓ. (The distance between two nodes is the length

[1] Note that this matrix does not have constant sparsity; however this can only increase the message complexity. Irregardless, it is possible to adapt the theoretical results to hold for this data as well, using techniques from [76].

[2] For the purposes of our simulations, such a permutation can be efficiently sampled, and guarantees perfect randomness. For larger n and m, it is possible to use pseudo-randomness with almost no loss in message complexity [76].



(a) (b) (c) (d)

Figure 5.1: Illustration of the number of messages required by ADMM and LOCO for the synthetic data set with results averaged over 50 trials. Plots (a) and (b) vary n while fixing sparsity $p = 10^{-4}$, showing the results in linear-scale and log-scale respectively. Plots (c) and (d) fix $n = 10^3$ and vary the sparsity p, showing the results in linear-scale and log-scale respectively.

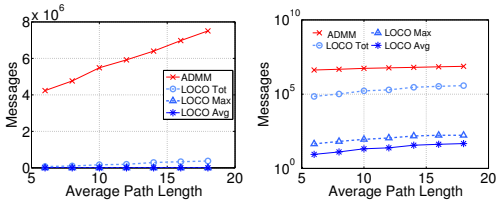
of the shortest path between them.) Then, we designate the path $L(i)$ to be the set of links comprising the shortest path between the source and the destination. The vectors c, $\underline{x}$, and $\bar{x}$ are chosen in the same manner as for the synthetic networks.

Algorithm tuning

Our results focus on comparing LOCO and ADMM. Running ADMM requires tuning four parameters [14]. Unless otherwise specified, we set the relative and absolute tolerances to be $\epsilon^{rel} = 10^{-4}$ and $\epsilon^{abs} = 10^{-2}$, the penalty parameter to be $\rho = 1$, and the maximum number of allowed iterations to be $t_{max} = 10000$. This is done to provide the best performance for ADMM: the parameters are tuned in the typical fashion to optimize ADMM [14]. Running LOCO requires tuning only one parameter: B, which governs the worst-case guarantee for the online algorithm used in step 2. A smaller B gives a "better guarantee", however some constraints may be violated. Setting $B = 2\ln(1+m)$ provides the best worst-case guarantee, and is our choice in the experiments unless stated otherwise. In fact, it is possible to tune B (akin to tuning ADMM) to specific data, as the constraints are often still satisfied for smaller B. In Figure 5.3 (c), we show the improvement in performance guarantee by tuning B, while keeping the dual solution feasible.

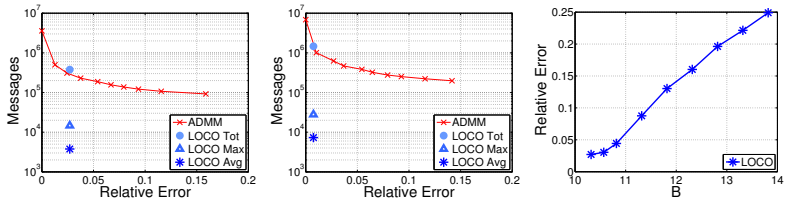
Metrics

For our numeric results, we evaluate ADMM and LOCO with respect to the quality of the solution provided and the number of messages sent.



(a) (b)

Figure 5.2: Illustration of the number of messages required by ADMM and LOCO for the real network data with $n = 8020$ and various average path lengths $L(i)$.



(a) (b) (c)

Figure 5.3: Comparison of the relative error and the number of messages required by LOCO and ADMM. Plots (a) and (b) show the Pareto optimal curve for ADMM with a range of relative tolerances $\epsilon^{rel} \in \{10^{-4}, 10^{-1}\}$. Plot (c) depicts how tuning B effects the relative error. The right most point corresponds to $B = 2\ln(1+m)$.

To assess the quality of the solution we measure the *relative error*, which is defined as $\frac{|p^* - p^{LOCO}|}{|p^*|}$, where p^* is the optimal solution. For problem instances of small dimension, one can run an interior point method to check the optimal solution, but this is too tedious for large problem sizes. In the large dimension cases we consider, we regard p^* to be ADMM's solution with small tolerances, such that the maximum number of allowed iterations is never needed. Note that the relative error is an empirical, normalized version of the approximation ratio for a given instance.

To measure the number of *messages* used by each of the algorithms, we consider the following. For a distributed implementation of ADMM, two sets of n variables are updated on separate processors and reported to a central controller which updates another variable (see [14, Chapter 7.1]). The number of *messages* for a run of ADMM is twice the number of sources in the NUM problem, multiplied by the number of iterations required by ADMM. LOCO needs to use communication only to construct the query set; running the online algorithm does not require any communication. Therefore, the number of messages

is proportional to the number of edges with at least one endpoint in the query set (this is the number of edges we need to send information over in order to construct the query set, see e.g., [76] for more details). We note that the number of messages depends both on the network topology and the realization of the ranking function.

5.2 Experimental Results

We now describe our empirical comparison of the performance of LOCO with ADMM.

Our first set of experiments investigates the communication used by ADMM and LOCO, i.e., the number of messages required. Figure 5.1 highlights that LOCO requires considerably fewer messages than ADMM, across both small and large n and varying levels of sparsity. More specifically, the figure shows that both the average and maximum amount of communication needed to answer a query about a specific piece of the solution under LOCO (LOCO Avg and LOCO Max respectively) are substantially lower than for ADMM. Further, even answering *every* query (LOCO Tot) requires only the same order of magnitude as ADMM. The figure includes ADMM with a tolerance ϵ^{rel} of 10^{-4} (ADMM 1) and 10^{-3} (ADMM 2). Even with suboptimal tolerance, which results in fewer iterations, ADMM still requires orders of magnitude more communication than LOCO.

Figure 5.2 shows the same qualitative behavior in the case of the real network data. In particular, the number of messages used is shown as a function of the average length of paths in the AS topology. We see that LOCO greatly outperforms ADMM for all tested average path lengths.

The improvement achieved by LOCO is possible because the size of the query sets used by LOCO are small compared to the number of sources. When $n = 10^3$, as in Figure 5.1, the number of nodes in the largest query set (over all trials) was 60.

We note that the improvement in the amount of communication is achieved at a cost: LOCO does not precisely solve the optimization, it only approximates the solution. When B is set to its worst-case guarantee (Figure 5.1), the relative error of LOCO ranges from 0.29 to 0.34.

It may seem somewhat unfair to compare the message complexity of LOCO and ADMM when they have differing relative error; we tune the parameters of

ADMM and LOCO such that the algorithms have comparable relative error, while LOCO Tot and ADMM require about the same number of messages. Figures 5.3 (a) and (b) illustrate the Pareto optimal frontier for ADMM: the minimal messages needed in order to obtain a particular relative error. Unlike ADMM, LOCO cannot trade off the number of messages used with the relative error, thus LOCO corresponds to a single point on the figures. This point is outside the Pareto frontier of ADMM. Figure 5.3 (c) illustrates the impact of tuning B. Similarly to ADMM, tuning B can significantly improve the relative error; unlike ADMM, tuning B does not affect the communication complexity.

Chapter 6

CONCLUDING REMARKS

We introduced a new, fundamentally different approach for distributed optimization based on techniques from the field of local computation algorithms. In particular, we designed a generic algorithm, LOCO, that constructs small neighborhoods and simulates an online algorithm on them. Due to the fact that LOCO is local, it has several advantages over existing methods for distributed optimization. In particular, it is more robust to network failures, communication lag, and changes in the system. To illustrate the benefits of LOCO we considered throughput maximization. The improvements of LOCO over ADMM in terms of communication in this setting are significant.

We view this work as a first step toward the investigation of local computation algorithms for distributed optimization. In future work, we intend to continue to investigate the performance of LOCO in more general network optimization problems. Further, it would be interesting to apply other techniques from the field of local computation algorithms to develop algorithms for other settings in which distributed computing is useful, such as power systems and machine learning.

Part III

Data Purchasing and Data Placement in a Geo-Distributed Data Market

Chapter 7

INTRODUCTION TO DISTRIBUTED DATA MARKETS

Ten years ago computing infrastructure was a *commodity* – the key bottleneck for new tech startups was the cost of acquiring and scaling computational power as they grew. Now, computing power and memory are *services* that can be cheaply subscribed to and scaled as needed via cloud providers like Amazon EC2, Microsoft Azure, etc.

We are beginning the same transition with respect to *data*. Data is broadly being gathered, bought, and sold in various marketplaces. However, it is still a commodity, often obtained through offline negotiations between providers and companies. Thus, acquiring data is one of the key bottlenecks for new tech startups nowadays.

This is beginning to change with the emergence of *cloud data markets*, which offer a single, logically centralized point for buying and selling data. Multiple data markets have recently emerged in the cloud, e.g., Microsoft Azure DataMarket [65], Factual [29], InfoChimps [44], Xignite [95], IUPHAR [88], etc. These marketplaces enable data providers to sell and upload data and clients to request data from multiple providers (often for a fee) through a unified query interface. They provide a variety of services: (i) *aggregation* of data from multiple sources, (ii) *cleaning* of data to ensure quality across sources, (iii) *ease of use*, through a unified API, and (iv) *low-latency delivery* through a geographically distributed content distribution network.

Given the recent emergence of data markets, there are widely differing designs in the marketplace today, especially with respect to pricing. For example, The Azure DataMarket [65] sets prices with a subscription model that allows a maximum number of queries (API calls) per month and limits the size of records that can be returned for a single query. Other data markets, e.g., Infochimps [44], allow payments per query or per data set. In nearly all cases, the data provider and the data market operator each then get a share of the fees paid by the clients, though how this share is arrived at can differ dramatically across data markets. The task of pricing is made even more challenging when one considers that clients may be interested in data with differing levels of

precision/quality and privacy may be a concern.

Not surprisingly, the design of pricing (both on the client side and the data provider side) has received significant attention in recent years, including pricing of per-query access [49, 51] and pricing of private data [32, 57].

In contrast, the focus of this paper is *not* on the design of pricing strategies for data markets. Instead, **we focus on the engineering side of the design of a data market**, which has been ignored to this point. Supposing that prices are given, there are important challenges that remain for the operation of a data market. Specifically, two crucial challenges relate to *data purchasing* and *data placement*.

Data purchasing: Given prices and contracts offered by data providers, which providers should a data market purchase from to satisfy a set of client queries with minimal cost?

Data placement: How should purchased data be stored and replicated throughout a geo-distributed data market in order to minimize bandwidth and latency costs? And which clients should be served from which replicas given the locations and data requirements of the clients?

Clearly, these two challenges are highly related: data placement decisions depend on which data is purchased from where, so the bandwidth and latency costs incurred because of data placement must be balanced against the purchasing costs. Concretely, less expensive data that results in larger bandwidth and latency costs is not desirable.

The goal of this work is to present a design for a geo-distributed data market that jointly optimizes data purchasing and data placement costs. The combination of data purchasing and data placement decisions makes the task of operating a geo-distributed data market more complex than the task of operating a geo-distributed data analytics system, which has received considerable attention in recent years e.g., [73, 93, 92]. Geo-analytics systems minimize the cost (in terms of latency and bandwidth) of moving the data needed to answer client queries, replacing the traditional operation mode where data from multiple data centers was moved to a central data center for processing queries. However, crucially, such systems do not consider the cost of obtaining the data (including purchasing and transferring) from data providers.

Thus, the design of a geo-distributed data market necessitates integrating data purchasing decisions into a geo-distributed data analytics system. To that end, our design builds on the model used in [92] by adding data providers that offer a menu of data quality levels for differing fees. The data placement/replication problem in [92] is already an integer linear program (ILP), and so it is no surprise that the addition of data providers makes the task of jointly optimizing data purchasing and data placement NP-hard (see Theorem 8).

Consequently, we focus on identifying structure in the problem that can allow for a practical and near-optimal system design. To that end, we show that the task of jointly optimizing data purchasing and data placement is equivalent to the uncapacitated facility location problem (UFLP) [52]. However, while constant-factor polynomial running time approximation algorithms are known for the *metric* uncapacitated facility location problem (see [17, 38, 45]), our problem is a *non-metric* facility location problem, and the best known polynomial running time algorithms achieve a $O(\log C)$ approximation via the greedy algorithm in [42] or the randomized rounding algorithm in [90], where C is the number of clients. Note that without any additional information on the costs, this approximation ratio is the smallest achievable for the non-metric uncapacitated facility location unless NP has slightly superpolynomial time algorithms [30]. While this is the best theoretical guarantee possible in the worst-case, some promising heuristics have been proposed for the non-metric case, e.g., [26, 8, 1, 48, 89, 36].

Though the task of jointly optimizing data purchasing and data placement is computationally hard in the worst case, in practical settings there is structure that can be exploited. In particular, we provide an algorithm with polynomial running time that gives an exact solution in the case of a data market with a single data center (§10.1). Then, using this structure, we generalize to the case of a geo-distributed data cloud and provide an algorithm, named Datum (§10.2) that is near optimal in practical settings.

Datum first optimizes data purchasing as if the data market was made up of a single data center (given carefully designed "transformed" costs) and then, given the data purchasing decisions, optimizes data placement/replication. The "transformed" costs are designed to allow an architectural decomposition of the joint problem into subproblems that manage data purchasing (external operations of the data market) and data placement (internal operations of

the data market). This decomposition is of crucial operational importance because it means that internal placement and routing decisions can proceed without factoring in data purchasing costs, mimicking operational structures of geo-distributed analytics systems today.

We provide a case study in §11 which highlights that Datum is near-optimal (within 1.6%) in practical settings. Further, the performance of Datum improves upon approaches that neglect data purchasing decisions by $> 45\%$.

To summarize, this paper makes the following main contributions:

1. We initiate the study of jointly optimizing data purchasing and data placement decisions in geo-distributed data markets.
2. We prove that the task of jointly optimizing data purchasing and data placement decisions is NP-hard and can be equivalently viewed as a facility location problem.
3. We provide an exact algorithm with polynomial running time for the case of a data market with a single data center.
4. We provide an algorithm, Datum, for jointly optimizing data purchasing and data placement in a geo-distributed data market that is within 1.6% of optimal in practical settings and improves by $> 45\%$ over designs that neglect data purchasing costs. Importantly, Datum decomposes into subproblems that manage data purchasing and data placement decisions separately.

Chapter 8

OPPORTUNITIES AND CHALLENGES

Data is now a traded *commodity*. It is being bought and sold every day, but most of these transactions still happen offline through direct negotiations for bulk purchases. This is beginning to change with the emergence of cloud data markets such as Microsoft Azure DataMarket in [65], Factual [29], InfoChimps [44], Xignite [95]. As cloud data markets become more prominent, data will become a *service* that can be acquired and scaled seamlessly, on demand, similarly to computing resources available today in the cloud.

8.1 The potential of data markets

The emergence of cloud data markets has the potential to be a significant disruptor for the tech industry, and beyond. Today, since computing resources can be easily obtained and scaled through cloud services, data acquisition has become the bottleneck for new tech startups.

For example, consider an emerging potential competitor for Yelp. The biggest development challenge is not algorithmic or computational. Instead, it is obtaining and managing high quality data at scale. The existence of a data market, e.g., Azure DataMarket, with detailed local information about restraints, attractions, etc., would eliminate this bottleneck entirely. In fact, data markets such as Factual [29] are emerging to target exactly this need.

Another example highlighted in [50, 51] is language translation. Emerging data markets such as the Azure DataMarket and Infochimps sell access to data on word translation, word frequency, etc. across languages. This access is a crucial tool for easing the transition tech startups face when moving into different cultural markets.

A final example considers computer vision. When tech startups need to develop computer vision tools in house, a significant bottleneck (in terms of time and cost) is obtaining labeled images with which to train new algorithms. Emerging data markets have the potential to eliminate this bottleneck too. For example, the emerging Visipedia project [91] (while free for now) provides an example of the potential of such a data market.

Thus, like in the case of cloud computing, ease of access and scaling, combined with the cost efficiency that comes with size, implies that cloud data markets have the potential to eliminate one of the major bottlenecks for tech startups today – data acquisition.

8.2 Operational challenges for data markets

The task of designing a cloud data market is complex, and requires balancing economic and engineering issues. It must carefully consider purchasing and pricing decisions in its interactions with both data providers and clients and minimize its operational cost, e.g., from bandwidth. We discuss both the economic and engineering design challenges below, though this paper focuses only on the engineering challenges.

Pricing

While there is a large body of literature on selling physical goods, the problem of pricing digital goods, such as data, is very different. Producing physical goods usually has a moderate fixed cost, for example, for buying the space and production machines needed, but this cost is partly recoverable: it is possible, if the company cannot manage to sell its product, to resell the machinery and buildings they have been using. However, the cost of producing and acquiring data is high and irrecoverable: if the data turns out to be worthless and nobody wants it, then the whole procedure is wasted. Another major difference comes from the fact that variable costs for data are low: once it has been produced, data can be cheaply copied and replicated.

These differences lead to "versioning" as the most typical approach for selling digital goods [7]. Versioning refers to selling different versions of the same digital good at different prices in order to target different types of buyers. This pricing model is common in the tech industry, e.g., companies like Dropbox sell digital space at different prices depending on how much space customers need and streaming websites such as Amazon often charge differently for streaming movies at different quality levels.

In the context of data markets, versioning is also common. For example, in Infochimps and the Azure DataMarket data consumers may pay a monthly subscription fee that varies according to the maximum number of queries they are allowed to run. Additionally, when charging per query, proposals have

suggested it is desirable to charge based on the complexity of the query, e.g., [6, 7]. Another form of versioning that has been proposed in data markets deals with privacy – data with more personal information should be charged more, e.g., [57, 22].

There is a growing literature focused on the design of pricing strategies for cloud data markets in the above, and other contexts, e.g., [6, 49, 7, 51, 7, 57, 22].

Data purchasing and data placement

While data pricing within cloud data markets has received increasing attention, the engineering of the system itself has been ignored. The engineering of such a geo-distributed "data cloud" is complex. In particular, the system must jointly make both data purchasing decisions and data placement, replication and delivery decisions, as described in the introduction.

Even considered independently, the task of optimizing data placement/replication within a geo-distributed data analytics system is challenging. Such systems aim to allow queries on databases that are stored across data centers, as opposed to traditional databased that are stored within a single data center. Examples include Google Spanner [21], Mesa [40], JetStream [74], Geode [92], and Iridium [73]. The aim in designing a geo-distributed data analytics system is to distribute the computation needed to answer queries across data centers; thus avoiding the need to transfer all the data to a single data center to respond to queries. This distribution of computation is crucial for minimizing bandwidth and latency costs, but leads to considerable engineering challenges, e.g., handling replication constraints due for fault tolerance and regulatory constraints on data placement due to data privacy. See [92, 73] for a longer discussion of these challenges and for examples illustrating the benefit of distributed query computation in geo-distributed data analytics systems.

Importantly, all previous work on geo-distributed analytics systems assumes that the system already owns the data. Thus, on top of the complexity in geo-distributed analytics systems, a geo-distributed cloud data market must balance the cost of data purchasing with the impact on data placement/replication costs as well as the decisions for data delivery. For example, if clients who are interested in some data are located close to data center A, while the data provider is located close to data center B (far from data center A), it may be

worth it to place that data in data center A rather than data center B. In practice, the problem is more complex since clients are usually geo-graphically distributed rather than centralized and one client may require data from several different data providers.

Additional complexity is created by versioning the data, i.e., the fact that clients have differing quality requirements for the data requested. For example, if some clients are interested in high quality data and others are interested in low quality data, then it may be worth it to provide high quality level data to some clients that only need low quality data (thus incurring a higher price) because of the savings in bandwidth and replication costs that result from being able to serve multiple clients with the same data.

Chapter 9

A GEO-DISTRIBUTED DATA CLOUD

This paper presents a design for a geo-distributed cloud data market, which we refer to as a "data cloud." This data cloud serves as an intermediary between *data providers*, which gather data and offer it for sale, and *clients*, which interact with the data cloud through queries for particular subsets/qualities of data. More concretely, the data cloud purchases data from multiple data providers, aggregates it, cleans it, stores it (across multiple geographically distributed data centers), and delivers it (with low-latency) to clients in response to queries, while aiming at minimizing the *operational cost* constituted of both bandwidth and data purchasing costs.

Our design builds on and extends the contributions of recent papers – specifically [92, 73] – that have focused on building geo-distributed data analytic systems but assume the data is already owned by the system and focus solely on the interaction between a data cloud and its clients. Unfortunately, as we highlight in §10, the inclusion of data providers means that the data cloud's goal of cost minimization can be viewed as a non-metric uncapacitated facility location problem, which is NP-hard.

For reference, Figure 9.1 provides an overview of the interaction between these three parties as well as some basic notations.

9.1 Modeling Data Providers

The interaction between the data cloud and data providers is a key distinction between the setting we consider and previous work on geo-distributed data analytics systems such as [73, 92]. We assume that each data provider offers distinct data to the data cloud, and that the data cloud is a price-taker, i.e., cannot impact the prices offered by data providers. Thus, we can summarize the interaction of a data provider with the data cloud through an exogenous menu of data qualities and corresponding prices.

We interpret the quality of data as a general concept that can be instantiated in multiple ways. For categorical data, quality may represent the resolution of the information provided, e.g., for geographical attributes the resolution may

be {street address, zip code, city, county, state}. For numerical data, quality could take many forms, e.g., the numerical precision, the statistical precision (e.g., the confidence of an estimator), or the level of noise added to the data.[1]

Concretely, we consider a setting where there are P data providers selling different data, $p \in \mathcal{P} = \{1, 2, \ldots, P\}$.[2] Each data provider offers a set of quality levels, indexed by $l \in \mathcal{L} = \{1, 2, \ldots, L_p\}$, where L_p is the number of levels that data provider p offers. We use $q(l, p)$ to denote the data quality level l, offered by data provider p. Similarly, we use $f(l, p)$ to denote the fee charged by data provider p for data of quality level l. Importantly, the prices vary across providers p since different providers have different procurement costs for different qualities and different data.

The data purchasing contract between data providers and data cloud may have a variety of different types. For example, a data cloud may pay data provider based on usage, i.e., per query, or a data cloud may buy the data in bulk in advance. In this paper, we discuss both per-query data contracting and bulk data contracting. See §9.3 for details.

9.2 Modeling Clients

Clients interact with the data cloud through queries, which may require data (with varying quality levels) from multiple data providers.

Concretely, we consider a setting where there are C clients, $c \in \mathcal{C} = \{1, 2, \ldots, C\}$. A client c sends a *query* to the data center, requesting particular data from multiple data providers.[3] Denote the set of data providers required by the request from client query c by $G(c)$. The client query also specifies a minimum desired quality level, $w_c(p)$, for each data provider p it requests, i.e., $\forall p \in G(c)$. We assume that the client is satisfied with data at a quality level higher than or equal to the level requested.

More general models of queries are possible, e.g., by including a DAG modeling the structure of the query and query execution planning (see [92] for details). For ease of exposition, we do not include such detailed structure here, but it

[1] A common suggestion for guaranteeing privacy is to add Laplace noise to data provided to data markets, see e.g., [25, 57]

[2] We distinguish data providers based on data, i.e., one data provider sells multiple data is treated as multiple data providers.

[3] We distinguish clients based on queries, i.e., one client sends multiple queries is treated as multiple clients.

can be added at the expense of more complicated notation.

Depending on the situation, the client may or may not be expected to pay the data cloud for access. If the clients are internal to the company running the data cloud, client payments are unnecessary. However, in many situations the client is expected to pay the data cloud for access to the data. There are many different types of payment structures that could be considered. Broadly, these fall into two categories: (i) subscription-based (e.g., Azure DataMarket [65]) or (ii) per-query-based (e.g. Infochimps [44]).

In this paper, we do not focus on (or model) the design of payment structure between the clients and the data cloud. Instead, we focus on the operational task of minimizing the cost of the data cloud operation (i.e., bandwidth and data purchasing costs). This focus is motivated by the fact that minimizing the operation costs improves the profit of the data cloud regardless of how clients are charged. Interested readers can find analyses of the design of client pricing strategies in [49, 51, 57].

9.3 Modeling a Geo-Distributed Data Cloud

The role of the data cloud in this marketplace is as an aggregator and intermediary. We model the data cloud as a geographically distributed cloud consisting of D data centers, $d \in \mathcal{D} = \{1, 2, \ldots, D\}$. Each data center aggregates data from geographically separate local data providers, and data from data providers may be (and often is) replicated across multiple data centers within the data cloud.

Note that, even for the same data with the same quality, data transfer from the data providers to the data cloud is not a one time event due to the need of the data providers to update the data over time. We target the modeling and optimization of data cloud within a fixed time horizon, given the assumption that queries from clients are known beforehand or can be predicted accurately. This assumption is consistent with previous work [92, 73] and reports from other organizations [94, 55]. Online versions of the problem are also of interest, but are not the focus of this paper.

Modeling costs

Our goal is to provide a design that minimizes the operational costs of a data cloud. These costs include both data purchasing and bandwidth costs. In or-

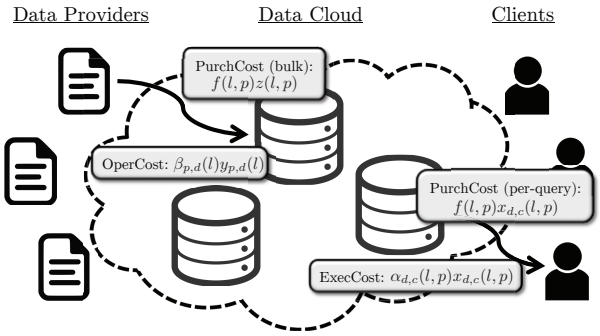


Figure 9.1: An overview of the interaction between data providers, the data cloud, and clients. The dotted line encircling the data centers (DC) represents the geo-distributed data cloud. Data providers and clients interact only with the cloud. Data provider p sends data of quality $q(l,p)$ to data center d, and the corresponding operation cost is $\beta_{p,d}(l)y_{p,d}(l)$. Similarly, data center d sends data of quality $q(l,p)$ to client c, and the corresponding execution cost is $\alpha_{d,c}(l,p)x_{d,c}(l,p)$. In bulk data contracting, the corresponding purchasing cost is $f(l,p)z(l,p)$. In per-query data contracting, the corresponding purchasing cost is $f(l,p)x_{d,c}(l,p)$.

der to describe these costs, we use the following notation, which is summarized in Figure 9.1.[4]

$x_{d,c}(l,p) \in \{0,1\}$: $x_{d,c}(l,p) = 1$ if and only if data of quality $q(l,p)$, originating from data provider p, is transferred from data center d to client c.

$\alpha_{d,c}(l,p)$: cost (including bandwidth and/or latency) to transfer data of quality $q(l,p)$, originating from data provider p, from data center d to client c

$y_{p,d}(l) \in \{0,1\}$: $y_{p,d}(l) = 1$ if and only if data of quality $q(l,p)$ is transferred from data provider p to data center d.

$\beta_{p,d}(l)$: cost (including bandwidth and/or latency) to transfer data of quality $q(l,p)$ from data provider p to data center d.

$z(l,p) \in \{0,1\}$: $z(l,p) = 1$ if and only if data of quality $q(l,p)$, originating from data provider p, is transferred to the data cloud.

[4]Throughout, subscript indices refer to data transfer "from, to" a location, and parenthesized indices refer to data characteristics (e.g., quality, from which data provider).

$f(l,p)$: purchasing cost of data with quality $q(l,p)$, originating from data provider p.

Given the above notations, the costs of the data cloud can be broken into three categories:

(i) The *operation cost* due to transferring data of all quality levels from data providers to data centers is

$$\text{OperCost} = \sum_{p=1}^{P} \sum_{l=1}^{L_p} \sum_{d=1}^{D} \beta_{p,d}(l) y_{p,d}(l). \tag{9.1}$$

(ii) The *execution cost* due to transferring data of all quality levels from data centers to clients is

$$\text{ExecCost} = \sum_{c=1}^{C} \sum_{p \in G(c)} \sum_{l=1}^{L_p} \sum_{d=1}^{D} \alpha_{d,c}(l,p) x_{d,c}(l,p). \tag{9.2}$$

(iii) The *purchasing cost* (PurchCost) due to buying data from the data provider could result from a variety of differing contract styles. In this paper we consider two extreme options: *per-query* and *bulk* data contracting. These are the most commonly adopted strategies for data purchasing today.

In *per-query* data contracting, the data provider charges the data cloud a fixed rate for each query that uses the data provided by the data provider. So, if the same data is used for two different queries, then the data cloud pays the data provider twice. Given a per-query fee $f(l,p)$ for data $q(l,p)$, the total purchasing cost is

$$\text{PurchCost(query)} = \sum_{c=1}^{C} \sum_{p \in G(c)} \sum_{l=1}^{L_p} \sum_{d=1}^{D} f(l,p) x_{d,c}(l,p). \tag{9.3}$$

In *bulk* data contracting, the data cloud purchases the data in bulk and then can distribute it without owing future payments to the data provider. Given a one-time fee $f(l,p)$ for data $q(l,p)$, the total purchasing cost is

$$\text{PurchCost(bulk)} = \sum_{p=1}^{P} \sum_{l=1}^{L_p} f(l,p) z(l,p). \tag{9.4}$$

To keep the presentation of the paper simple, we focus on the per-query data contracting model throughout the body of the paper and discuss the bulk data contracting model (which is simpler) in Appendix C.3.

Cost Optimization

Given the cost models described above, we can now represent the goal of the data cloud via the following integer linear program (ILP), where OperCost, ExecCost, and PurchCost are as described in equations (9.1), (9.2) and (9.3), respectively.

$$\min_{x,y} \quad \text{OperCost} + \text{ExecCost} + \text{PurchCost} \tag{9.5}$$

$$\text{subject to} \quad x_{d,c}(l,p) \leq y_{p,d}(l)\ \forall c,p,l,d \tag{9.5a}$$

$$\sum_{l=1}^{L_p}\sum_{d=1}^{D} x_{d,c}(l,p) = 1,\ \forall c, p \in G(c) \tag{9.5b}$$

$$\sum_{l=1}^{L_p}\sum_{d=1}^{D} x_{d,c}(l,p)q(l,p) \geq w_c(p),\ \forall c, p \in G(c) \tag{9.5c}$$

$$x_{d,c}(l,p) \geq 0, \forall c,p,l,d \tag{9.5d}$$

$$y_{p,d}(l) \geq 0,\ \forall p,l,d \tag{9.5e}$$

$$x_{d,c}(l,p), y_{p,d}(l) \in \{0,1\}, \forall c,p,l,d \tag{9.5f}$$

The constraints in this formulation warrant some discussion. Constraint (9.5a) states that any data transferred to some client must already have been transferred from its data provider to the data cloud.[5] Constraint (9.5b) ensures that each client must get the data it requested, and constraint (9.5c) ensures that the minimum quality requirement of each client must be satisfied. The remaining constraints state that the decision variables are binary and nonnegative.

An important observation about the formulation above is that data purchasing/placement decisions are decoupled across data providers, i.e., the data purchasing/placement decision for data from one data provider does not impact the data purchasing/placement decision for any other data providers. Thus, we frequently drop the index p.

[5]For bulk data contracting model, one more constraint $y_{p,d}(l) \leq z(l,p),\ \forall c,l,p,d$ is required. This constraint states that any data placed in the data cloud must be purchased by the data cloud.

Note that there are a variety of practical issues that we have not incorporated into the formulation in (9.5) in order to minimize notational complexity, but which can be included without affecting the results described in the following. A first example is that a minimal level of data replication is often desired for fault tolerance and disaster recovery reasons. This can be added to (9.5) by additionally considering constraints of the form $\sum_{d=1}^{D} y_{p,d}(l) \geq kz(l,p)$, where k denotes the minimum required number of copies. Similarly, privacy concerns often lead to regulatory constraints on data movement. As a result, regulatory restrictions may prohibit some data from being copied to certain data centers, thus constraining data placement and replication. This can be included by adding constraints of the form $y_{p,d}(l) = 0$ to (9.5) where p and d denote the corresponding data provider and data center, respectively. Finally, in some cases it is desirable to enforce SLA constraints on the latency of delivery to clients. Such constraints can be added by including constraints of the form $\sum_{p \in G(c)} \sum_{l=1}^{L_p} \sum_{d=1}^{D} \alpha_{d,c}(l,p) x_{d,c}(l,p) \leq r_c$, where r_c denotes the SLA requirement of client c.

We refer the reader to [92, 93, 73] for more discussions of these additional practical constraints. Each paper includes a subset of these factors in the design of geo-distributed data analytics systems, but does not model data purchasing decisions.

Chapter 10

OPTIMAL DATA PURCHASING & DATA PLACEMENT

Given the model of a geo-distributed data cloud described in the previous section, the design task is now to provide an algorithm for computing the optimal data purchasing and data placement/replication decisions, i.e., to solve data cloud cost minimization problem in (9.5). Unfortunately, this cost minimization problem is an ILP, which are computationally difficult in general.[1]

A classic NP-hard ILP is the uncapacitated facility location problem (UFLP) [52]. In the uncapacitated facility location problem, there is a set of I clients and J potential facilities. Facility $j \in J$ costs f_j to open and can serve clients $i \in I$ with cost $c_{i,j}$. The task is to determine the set of facilities that serves the clients with minimal cost.

Our first result, stated below, highlights that cost minimization for a geo-distributed data cloud can be reduced to the uncapacitated facility location problem, and vice-versa. Thus, the task of operating a data cloud can then be viewed as a facility location problem, where opening a facility parallels purchasing a specific quality level from a data provider and placing it in a particular data center in the data cloud.

Theorem 8 *The cost minimization problem for a geo-distributed data cloud given in* (9.5) *is NP-hard.*

The proof of Theorem 8 (given in Appendix C) provides a reduction both to and from the uncapacitated facility location problem. Importantly, the proof of Theorem 8 serves a dual purpose: it both characterizes the hardness of the data cloud cost minimization problem and highlights that algorithms for the facility location problem can be applied in this context. Given the large literature on facility location, this is important.

More specifically, the reduction leading to Theorem 8 highlights that the data cloud optimization problem is equivalent to the *non-metric* uncapacitated fa-

[1]Note that previous work on geo-distributed data analytics where data providers and data purchasing were not considered already leads to an ILP with limited structure. For example, [92] suggest only heuristic algorithms with no analytic guarantees.

cility location problem – every instance of any of the two problems can be written as an instance of the other. While constant-factor polynomial running time approximation algorithms are given for the *metric* uncapacitated facility location problem in [17, 38, 45], in the more general *non-metric* case the best known polynomial running time algorithm achieves a $\log(C)$-approximation via a greedy algorithm with polynomial running time, where C is the number of clients [42]. This is the best worst-case guarantee possible (unless NP has slightly superpolynomial time algorithms, as proven in [30]); however some promising heuristics have been proposed for the non-metric case, e.g., [26, 8, 1, 48, 89, 36].

Nevertheless, even though our problem can, in general, be viewed as the non-metric uncapacitated facility location, it does have a structure in real-world situations that we can exploit to develop practical algorithms.

In particular, in this section we begin with the case of a data cloud made up of a single data center. We show that, in this case, there is a structure that allows us to design an algorithm with polynomial running time that gives an exact solution (§10.1). Then, we move to the case of a data cloud made up of geo-distributed data centers and highlight how to build on the algorithm for the single data center case to provide an algorithm, Datum, for the general case (§10.2). Importantly, Datum allows decomposition of the management of data purchasing (operations outside of the data cloud) and data placement (operations inside the data cloud). This feature of Datum is crucial in practice because it means that the algorithm allows a data cloud to manage internal operations without factoring in data purchasing costs, mimicking operations today. While we do not provide analytic guarantees for Datum (as expected given the reduction to/from the non-metric facility location problem), we show that the heuristic performs well in practical settings using a case study in §11.

10.1 An exact solution for a single data center

We begin our analysis by focusing on the case of a single data center, which interacts with multiple data providers and multiple clients. The key observation is that, if the execution costs associated with transferring different quality levels of the same data are the same, i.e., $\forall l, \alpha_c(l) = \alpha_c$, then the execution cost becomes a constant which is independent of the data purchasing and data

placement decisions as shown in (10.1).

$$\text{ExecCost} = \sum_{c=1}^{C} \sum_{l=1}^{L} \alpha_c x_c(l) = \sum_{c=1}^{C} \alpha_c \left(\sum_{l=1}^{L} x_c(l) \right) = \sum_{c=1}^{C} \alpha_c \tag{10.1}$$

The assumption that the execution costs are the same across quality levels is natural in many cases. For example, if quality levels correspond to the level of noise added to numerical data, then the size of the data sets will be the same. We adopt this assumption in what follows.

This assumption allows the elimination of the execution cost term from the objective. Additionally, we can simplify notation by removing the index d for the data center. Thus, in per-query data contracting, the data cloud optimization problem can be simplified to (10.2). (We discuss the case of bulk data contracting in Appendix C.3.)

$$\begin{aligned}
\text{minimize} \quad & \sum_{l=1}^{L} \beta(l) y(l) + \sum_{c=1}^{C} \sum_{l=1}^{L} f(l) x_c(l) && (10.2)\\
\text{subject to} \quad & x_c(l) \leq y(l), \ \forall c, l \\
& \sum_{l=w_c}^{L} x_c(l) = 1, \ \forall c && (10.2a)\\
& x_c(l) \geq 0, \forall c, l \\
& y(l) \geq 0, \ \forall l \\
& x_c(l), y(l) \in \{0, 1\}, \forall c, l
\end{aligned}$$

Note that constraint (10.2a) is a contraction of (9.5b) and (9.5c), and simply means that any client c must be given exactly one quality level above w_c, the minimum required quality level.[2] While this problem is still an ILP, in this case there is a structure that can be exploited to provide a polynomial time algorithm that can find an exact solution. In particular, we prove in Appendix C.1 that the solution to (10.2) can be found by solving the linear program (LP) given in (10.3).

[2]While the two constraints are equivalent for an ILP, they lead to different feasible sets when considering its LP-relaxation; in particular, facility location algorithms based on LP-relaxations such as randomized rounding algorithms need to use the contracted version of the constraints to preserve the $O(\log C)$-approximation ratio for non-metric facility location. It is equivalent to the reformulation given in Appendix C and does not introduce infinite costs that may lead to numerical errors.

$$\begin{aligned} \text{minimize} \quad & \sum_{l=1}^{L} \beta(l) y(l) + \sum_{i=1}^{L} \sum_{l=i}^{L} S_i f(l) \chi_i(l) \qquad (10.3) \\ \text{subject to} \quad & \\ & \chi_i(l) \leq y(l), \ \forall i, l \\ & \sum_{l=i}^{L} \chi_i(l) = 1, \ \forall i \\ & \chi_i(l) \geq 0, \forall i, l \\ & y(l) \geq 0, \ \forall l \end{aligned}$$

In (10.3), S_i is the number of clients who require a minimum quality level of i, and $\chi_i(l) = 1$ represents clients with minimum required quality level i purchase at quality level l.

Note that this LP is not directly obtained by relaxing the integer constraints in (10.2), but is obtained from relaxing the integer constraints in a reformulation of (10.2) described in Appendix C.1. The theorem below provides a tractable, exact algorithm for cost minimization in a data cloud made up of a single data center. (A proof is given in Appendix C.1).

Theorem 9 *There exists a binary optimal solution to the linear relaxiation program in* (10.3) *which is an optimal solution of the integer program in* (10.2) *and can be found in polynomial time.*

In summary, the following gives a polynomial time algorithm which yields the optimal solution of (10.2).

Step 1: Rewrite (10.2) in the form given by (C.4).

Step 2: Solve the linear relaxation of (C.4), i.e., (10.3). If it gives an integral solution, this solution is an optimal solution of (10.2), and the algorithm finishes. Otherwise, denote the fractional solution of the previous step by $\{\chi^r(l), y^r(l)\}$ and continue to the next step.

Step 3: Find $m_i \in \{i, \ldots, n\}$ such that $\sum_{l=i}^{m_i - 1} y^r(l) < 1$, and $\sum_{l=i}^{m_i} y^r(l) \geq 1$. (See Appendix C.1 for the existence of $\{m_i\}$.) And express $\{\chi_i(l)\}$ as a function of $\{y(l)\}$ based on (C.6). Substitute the expressions of $\{\chi_i(l)\}$ with $\{y(l)\}$

in (10.3) to obtain an instance of (C.7). Solve the linear programming problem (C.7) and find an optimal solution that is also an extreme point of (C.7).[3] This yields a binary optimal solution of (C.7). Use transformation (C.6) to get a binary optimal solution of (10.3), which can be reformulated as an optimal solution of (10.2) from the definition of $\{\chi_i(l)\}$.

10.2 The design of Datum

Unlike the data cloud cost minimization problem for a single data center, the general data cloud cost minimization is NP-hard. In this section, we build on the exact algorithm for cost minimization in a data cloud made up of a single data center (§10.1) to provide an algorithm, Datum, for cost minimization in a geo-distributed data cloud.

The idea underlying Datum is to, first, optimize data purchasing decisions as if the data market was made up of a single data center (given carefully designed "transformed" costs), which can be done tractably as a result of Theorem 9. Then, second, Datum optimizes data placement/replication decisions given the data purchasing decisions.

Before presenting Datum, we need to reformulate the general cost minimization ILP in (9.5). Recall that (9.5) is separable across providers, thus we can consider independent optimizations for each provider, and drop the index p throughout. Second, we denote the set of all possible subsets of data centers, e.g., $\{\{d_1\}, \{d_2\}, \ldots, \{d_1, d_2\}, \{d_1, d_3\}, \ldots\}$ by V.[4] Further, define $\beta_v(l) = \sum_{d \in v} \beta_d(l)$, and $\alpha_{v,c}(l) = \min_{d \in v}\{\alpha_{d,c}(l)\}$. Given this change, we define $y_v(l) = 1$ if and only if data with quality level l is placed in (and only in) data centers $d \in v$ and $x_{v,c}(l) = 1$ if and only if data with quality level l is transferred to client c from some data center $d \in v$. These reformulations allow us to convert (9.5) to (10.4) as following.

[3] This step can be finished in polynomial time [11].

[4] Note that, in practice, the number of data centers is usually small, e.g., $10 - 20$ worldwide. Further, to avoid exponential explosion of V, the subsets included in V can be limited to only have a constant number of data centers, where the constant is determined by the maximal number of replicas to be stored.

$$\text{minimize} \quad \sum_{l=1}^{L}\sum_{v=1}^{V}\beta_v(l)y_v(l) + \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{v=1}^{V}\alpha_{v,c}(l)x_{v,c}(l) + \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{v=1}^{V}f(l)x_{v,c}(l) \tag{10.4}$$

$$\text{subject to} \quad x_{v,c}(l) \leq y_v(l),\ \forall c, l \tag{10.4a}$$

$$\sum_{l=w_c}^{L}\sum_{v=1}^{V}x_{v,c}(l) = 1,\ \forall c \tag{10.4b}$$

$$\sum_{v=1}^{V}y_v(l) \leq 1,\ \forall l \tag{10.4c}$$

$$\sum_{v=1}^{V}x_{v,c}(l) \leq 1,\ \forall c, l \tag{10.4d}$$

$$x_{v,c}(l) \geq 0, \forall v, c, l \tag{10.4e}$$

$$y_v(l) \geq 0,\ \forall v, l \tag{10.4f}$$

$$x_{v,c}(l), y_v(l) \in \{0, 1\}, \forall v, c, l \tag{10.4g}$$

Compared to (9.5), the main difference is that (10.4) has two extra constraints (10.4c) and (10.4d). Constraint (10.4c) ensures that data can only be placed in at most one subset of data centers across V. And constraint (10.4d) follows from constraint (10.4b). Using this reformulation Datum can now be explained in two steps.

Step 1: Solve (10.5) while treating the geo-distributed data cloud as a single data center. Specifically, define $Y(l) = \sum_{v=1}^{V} y_v(l)$ and $X_c(l) = \sum_{v=1}^{V} x_{v,c}(l)$. Note that, $Y(l)$ and $X_c(l)$ are $0-1$ variables from Constaint (10.4c) and (10.4d). Further, ignore the middle term in the objective, i.e., the ExecCost. Finally, for each quality level l, consider a "transformed" cost $\beta^*(l)$. We discuss how to define $\beta^*(l)$ below. This leaves the "single data center" problem (10.5). Crucially, this formulation can be solved optimally in polynomial time using the results for the case of a data cloud made up of a single data center (§10.1).

$$
\begin{aligned}
\text{minimize} \quad & \sum_{l=1}^{L} \beta^*(l) Y(l) + \sum_{c=1}^{C} \sum_{l=1}^{L} f(l) X_c(l) && (10.5)\\
\text{subject to} \quad & X_c(l) \le Y(l),\ \forall c, l \\
& \sum_{l=w_c}^{L} X_c(l) = 1,\ \forall c \\
& X_c(l) \ge 0, \forall c, l \\
& Y(l) \ge 0,\ \forall l \\
& X_c(l), Y(l) \in \{0,1\}, \forall c, l
\end{aligned}
$$

The remaining issue is to define $\beta^*(l)$. Note that the reason for using transformed costs $\beta^*(l)$ instead of $\beta_v(l)$ is that $\beta_v(l)$ cannot be known precisely without also optimizing the data placement. Thus, in defining $\beta^*(l)$ we need to anticipate the execution costs that result from data placement and replication given the purchase of data with quality level l. This anticipation then allows a decomposition of data purchasing and data placement decisions. Note that the only inaccuracy in the heuristic comes from the mismatch between $\beta^*(l)$ and $\min\{\beta_v(l) + \sum_{c \in C^*(l)} \alpha_{v,c}(l)\}$ where $C^*(l)$ is the set of customers who buy at quality level l in an optimal solution – if these match for the minimizer of (9.5) then the heuristic is exact. Indeed, in order to minimize the cost of locating quality levels to data centers, and allocating clients to data centers and quality levels, the set of data centers v where an optimal solution chooses to put quality level l has to minimize the cost of data transfer in the set v and allocating all clients who get data at quality level l, i.e. $C^*(l)$, to this set of data centers v.

Many choices are possible for the transformed costs $\beta^*(l)$. A conservative choice is $\beta^*(l) = \min_v \beta_v(l)$, which results in a solution (with Step 2) whose OperCost + PurchCost is a lower bound to the corresponding costs in the optimal solution of (9.5).[5] However, it is natural to think that more aggressive estimates may be valuable. To evaluate this, we have performed experiments in the setting of the case study (see §11) using the following parametric form $\beta^*(l) = \min_v \{\beta_v(l) + \mu_1 \sum_{l' \le l} \sum_{w_c = l'} \alpha_{v,c}(l') e^{-\mu_2 (l - l')}\}$, where μ_1 and μ_2 are parameters. This

[5] However the ExecCost cannot be bounded, thus we cannot obtain a bound for the total cost. The proof of this is simple and is not included in the paper due to space limit.

form generalizes the conservative choice by providing a weighting of $\alpha_{v,c}(l')$ based on the "distance" of the quality deviation between l' and the target quality level l. The idea behind this is that a client is more likely to be served data with quality level close to the requested minimum quality level of the client. Here we use the exponential decay term $e^{-\mu_2(l-l')}$ to capture the possibility of serving the data with quality level l to a client with minimum quality level $l' \leq l$. Interestingly, in the setting of our case study, the best design is $\mu_1 = \mu_2 = 0$, i.e., the conservative estimate $\beta^*(l) = \min_v \beta_v(l)$, and so we adopt this $\beta^*(l)$ in Datum.

Step 2: At the completion of Step 1 the solution (X, Y) to (10.5) determines which quality levels should be purchased and which quality level should be delivered to each client. What remains is to determine data placement and data replication levels. To accomplish this, we substitute (X, Y) into (10.4), which yields (10.6).

$$\begin{aligned}
\text{minimize} \quad & \sum_{l=1}^{L}\sum_{v=1}^{V} \beta_v(l) y_v(l) + \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{v=1}^{V} \alpha_{v,c}(l) x_{v,c}(l) \\
& + \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{v=1}^{V} f(l) x_{v,c}(l) && (10.6) \\
\text{subject to} \quad & x_{v,c}(l) \leq y_v(l), \ \forall c, l && (10.6a) \\
& \sum_{l=w_c}^{L}\sum_{v=1}^{V} x_{v,c}(l) = 1, \ \forall c && (10.6b) \\
& \sum_{v=1}^{V} y_v(l) = Y(l) && (10.6c) \\
& \sum_{v=1}^{V} x_{v,c}(l) = X_v(l) && (10.6d) \\
& x_{v,c}(l) \geq 0, \forall v, c, l && (10.6e) \\
& y_v(l) \geq 0, \ \forall v, l && (10.6f) \\
& x_{v,c}(l), y_v(l) \in \{0, 1\}, \forall v, c, l && (10.6g)
\end{aligned}$$

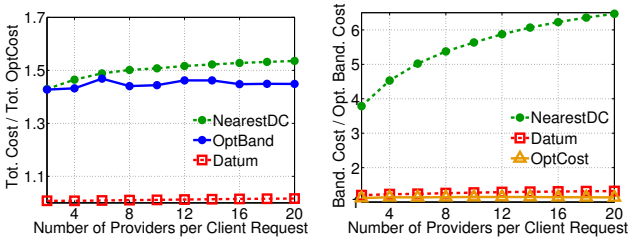
The key observation is that this is no longer a computationally hard ILP. In fact, the inclusion of (X, Y) means that it can be solved in closed form.

Let $C(l)$ denote the set of clients that purchase data with quality level l, i.e.,

$C(l) = \{c : X_c(l) = 1\}$. Then (10.7) gives the optimal solution of (10.6). (A proof is given in Appendix C.2.)

$$y_v(l) = \begin{cases} 1, & \text{if } Y(l) = 1 \text{ and} \\ & v = \arg\min\{\beta_v(l) + \sum_{c \in C(l)} \alpha_{v,c}(l)\}, \\ 0, & \text{otherwise.} \end{cases} \tag{10.7a}$$

$$x_{v,c}(l) = \begin{cases} y_v(l), & \text{if } c \in C(l), \\ 0, & \text{otherwise.} \end{cases} \tag{10.7b}$$



(a) Total Cost (b) Bandwidth Cost

Figure 11.1: Illustration of the near-optimality of Datum as a function of the complexity of client requests (i.e., the average number of providers data must be procured from in order to complete a client request).

Chapter 11

CASE STUDY

We now illustrate the performance of Datum using a case study of a geo-distributed data cloud running in North America. While the setting we use is synthetic, we attempt to faithfully model realistic geography for data centers in the data cloud, data providers, and clients. Our focus is on quantifying the overall cost (including data purchasing and bandwidth/latency costs) of Datum compared to two existing designs for geo-distributed data analytics systems and the optimal. To summarize, the highlights of our analysis are

1. Datum provides consistently lower cost ($> 45\%$ lower) than existing designs for geo-distributed data analytics systems.
2. Datum achieves near optimal total cost (within 1.6%) of optimal.
3. Datum achieves reduction in total cost by significantly lowering purchasing costs without sacrificing bandwidth/latency costs, which stay typically within 20-25% of the minimal bandwidth/latency costs necessary for delivery of the data to clients.

11.1 Experimental setup

The following outlines the setting in which we demonstrate the empirical performance of Datum.

Geo-distributed data cloud. We consider a geographically distributed data cloud

with 10 data centers located in California, Washington, Oregon, Illinois, Georgia, Virginia, Texas, Florida, North Carolina, and South Carolina. The locations of the data centers in our experiments mimic those in [37] and include the locations of Google's data centers in the United States.

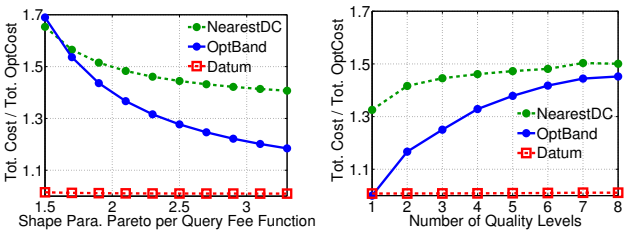
Clients. Client locations are picked randomly among US cities, weighted proportionally to city populations. Each client requests data from a subset of data providers, chosen *i.i.d.* from a Uniform distribution. Unless otherwise specified, the average number of providers per client request is $P/2$. The quality level requested from each chosen provider follows a Zipf distribution with mean $L_p/2$ and shape parameter 30. P and L_p are defined as in §9.1 and §9.2. We choose a Zipf distribution motivated by the fact that popularity typically follows a heavy-tailed distribution [68]. Results are averaged over 20 random instances. We observe that the results of the 20 instances for the same plot are very close (within 5%), and thus do not show the confidence intervals on the plots.

Data providers. We consider 20 data providers. We place data providers in the second and third largest cities within a state containing a data center. This ensures that the data providers are near by, but not right on top of, data center and client locations.

Operation and execution costs. To set operation and execution costs, we compute the geographical distances between data centers, clients and providers. The operation and execution costs are proportional to the geographical distances, such that the costs are effectively one dollar per gigameter. This captures both the form of bandwidth costs adopted in [93] and the form of latency costs adopted in [73].

Data purchasing costs. The per-query purchasing costs are drawn *i.i.d.* from a Pareto distribution with mean 10 and shape parameter 2 unless otherwise specified. We choose a Pareto distribution motivated by the fact that incomes and prices often follow heavy-tailed distributions [68]. Results were averaged over 20 random instances. To study the sensitivity of Datum to the relative size of purchasing and bandwidth costs, we vary the ratio of them between $(0.01, 100)$.

Baselines. We compare the performance of Datum to the following baselines.



(a) (b)

Figure 11.2: Illustration of Datum's sensitivity to query parameters. (a) varies the heaviness of the tail in the distribution of purchasing fees. (b) varies the number of quality levels available. Note that Figure 11.1 sets the shape parameter of the Pareto governing purchasing fees to 2 and includes 8 quality levels.

- *OptCost* computes the optimal solution to the data cloud cost minimization problem by solving the integer linear programming (9.5). Note that this requires solving an NP-hard problem, and so is not feasible in practice. We include it in order to benchmark the performance of Datum.

- *OptBand* computes the optimal solution to the bandwidth cost minimization problem. It is obtained by minimizing only the operation cost and execution cost in the objective of (9.5). Bandwidth cost minimization is commonly considered as a primary goal for cost minimization in geo-distributed data analytics systems [92]. Due to computational complexity, heuristics are usually applied to minimize the bandwidth cost. Here, instead of implementing a heuristic algorithms, we optimistically use OptBand in order to lower bound the achievable performance. Note that this also requires solving an NP-hard problem and thus is not feasible in practice.

- *NearestDC* is a greedy heuristic for the total cost minimization problem that is often applied in practice. It serves the clients exactly what they ask for by purchasing the data and storing it at the data center closest to the data provider.

11.2 Experimental results

Quantifying cost reductions from Datum. Figure 11.1(a) illustrates the costs savings Datum provides. Across levels of query complexity (number of providers involved), Datum consistently provides $> 45\%$ savings over OptBand and $> 51\%$ savings compared to NearestDC. Further, Datum is within 1.6% of

the optimal cost in all these cases. The improvement of Datum compared to OptBand comes as a result of optimizing purchasing decisions at the expense of increased bandwidth. Importantly, Figure 11.1(b) shows that the extra bandwidth cost incurred is small, $20 - 25\%$. Thus, joint optimization of data purchasing and data placement decisions leads to significant reductions in total cost without adversely impacting bandwidth costs.

The form of client queries. To understand the sensitivity of the cost reductions provided by Datum, we next consider the impact of parameters related to client queries. Figure 11.1 shows that the complexity of queries has little impact on the cost reductions of Datum. Figure 11.2 studies two other parameters: the heaviness of the tail of the per-query purchasing fee and the number of quality levels offered.

Across all settings, Datum is within 1.6% of optimal; however both of these parameters have a considerable impact on the cost savings Datum provides over our baselines. In particular, the lighter the tail of the prices of different quality levels is, the less improvement can be achieved. This is a result of more concentration of prices across quality levels leaving less room for optimization. Similarly, fewer quality levels provides less opportunity to optimize data purchasing decisions. At the extreme, with only quality level available, the opportunity to optimization data purchasing goes away and OptBand and OptCost are equivalent.

Data purchasing vs. bandwidth costs. The most important determinant of the magnitude of Datum's cost savings is the relative importance of data purchasing costs. In one extreme, if data is free, then the data purchasing decisions disappear and the problem is simply to do data placement in a manner that minimizes bandwidth costs. In the other extreme, if data purchasing costs dominate then data placement is unimportant. In Figure 11.3 we only compare total costs among OptCost, OptBand, and Datum. NearestDC is far worse (more than 5 times worse than OptCost in some cases) and thus is dropped from the plots. Figure 11.3(a) studies the impact of the relative size of data purchasing and bandwidth costs. When the x-axis is 0, the data purchasing and bandwidth costs of the data center are balanced. Positive values mean that bandwidth costs dominate and negative values mean that data purchasing costs dominate. As expected, Datum's cost savings are most dramatic in regimes where data purchasing costs dominate. Cost savings can be

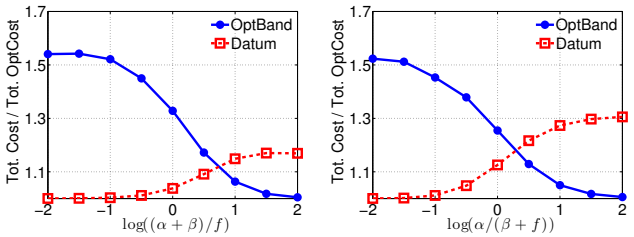


Figure 11.3: Illustration of the impact of bandwidth and purchasing fees on Datum's performance. NearestDC is excluded because its costs are off-scale. (a) varies the ratio of bandwidth costs (summarized by $\alpha + \beta$) to purchasing costs (summarized by f). (b) varies the ratio of costs internal to the data cloud (α) to costs external to the data cloud ($\beta + f$). Note that in Figure 11.1 the ratios are set to $\log(\frac{\alpha+\beta}{f}) = -0.5$ and $\log(\frac{\alpha}{\beta+f}) = -1$.

54% in extreme settings. Data purchasing costs are expected to dominate in the future – for some systems this is already true today. However, it is worth noting that, in settings where bandwidth costs dominates, Datum can deviate from the optimal cost by $10 - 20\%$ in extreme circumstances, and can be outperformed by the MinBand benchmark. Of course, Datum is not designed for such settings given its prioritization of the minimization of data purchasing costs.

Internal vs. external costs. An important aspect of the design of Datum is the decomposition of data purchasing decisions from data placement decisions. This provides a separation between the internal and external operations (and costs) of Datum. Given this separation, it is important to evaluate the sensitivity of Datum's design to the relative size of internal and external costs.

Given that Datum prioritizes the optimization of external costs (optimizing them in Step 1, see §10.2), it is natural to expect that Datum performs best when these costs dominate. This is indeed the case, as illustrated in Figure 11.3(b). Like in Figure 11.3(a), when the x-axis is 0, the internal and external costs are balanced. Positive values indicate the internal costs dominate and negative values indicate the external costs dominate. In settings where external costs dominate Datum can provide 50% cost savings and be within a few percent of the optimal. However, in cases when internal costs dominate Datum can deviate from the optimal cost by $10 - 30\%$ in extreme circumstances, and can be outperformed by the MinBand benchmark. Note that, as data purchasing costs grow in importance, external costs will dominate, and so

we can expect that Datum will provide near optimal performance in practical settings.

Chapter 12

RELATED WORK

Our work focuses on the joint design of data purchasing and data placement in a geo-distributed cloud data market. As such, it is related both to recent work on data pricing and to geo-distributed data analytics systems. Further, the algorithmic problem at the core of our design is the facility location problem, and so our work builds on that literature. We discuss related work in these three areas in the following.

Data pricing: The design of data markets has begun to attract increasing interest in recent years, especially in the database community, see [6] for an overview. The current literature mainly focuses on query-based pricing mechanism designs [49, 51, 57] and seldom considers the operating cost of the market service providers (i.e., the data cloud). There is also a growing body of work related to data pricing with differentiated qualities [32, 57, 22], often motivated by privacy. See §8.2 for more discussion. This work relates to data pricing on the data provider side and is orthogonal to our discussion in this paper.

Geo-distributed data analytics systems: As cloud servers are increasingly located in geo-distributed systems, analysis and optimization of data stored in geographically distributed data centers has received increasing attention [92, 93, 73, 43]. Bandwidth constraints [92, 93] as well as latency [73] are the two main challenges for system design, and a number of system designs have been proposed, e.g., see §8.2 for more discussion. Our work builds on the model of geo-distributed data analytics systems in [73, 92], but is distinct from this literature because none of the work on geo-distributed data analytics systems considers the costs associated with purchasing data.

Algorithms for facility location: Our data cloud cost minimization problem can be viewed as a variant of the uncapacitated facility location problem. Though such problems have been widely studied, most of the results, especially algorithms with constant approximation ratios, require the assumption of metric cost parameters [17, 38, 45], which is not the case in our problem. In contrast, for the non-metric facility location problem the best known algorithm is a greedy algorithm proposed in [52]. Beyond this algorithm, a

variety of heuristics have been proposed, however none of the heuristics are appealing for our problem because it is desirable to separate (external) data purchasing decisions from (internal) data placement/replication decisions as much as possible. As a result we propose a new algorithm, Datum, which is both near-optimal in practical settings and provides the desired decomposition. Datum may also be valuable more broadly for facility location problems.

Chapter 13

CONCLUSION

This work sits at the intersection of two recent trends: the emergence of online data marketplaces and the emergence of geo-distributed data analytics systems. Both have received significant attention in recent years across academia and industry, changing the way data is bought and sold and changing how companies like Facebook run queries across geo-distributed databases. In this paper we study the engineering challenges that come when online data marketplaces are run on top of a geo-distributed data analytics infrastructure. Such cloud data markets have the potential to be a significant disruptor (as we highlight in §8). However, there are many unanswered economic and engineering questions about their design. While there has been significant prior work on economic questions, such as how to price data, the engineering questions have been neglected to this point.

We presented the design of a geo-distributed cloud data market: Datum. Datum jointly optimizes data purchasing decisions with data placement decisions in order to minimize the overall cost. While the overall cost minimization problem is NP-hard (via a reduction to/from the facility location problem), Datum provides near-optimal performance (within 1.6% of optimal) in realistic settings via a polynomial-time algorithm that is provably optimal in the case of a data cloud running on a single data center. Additionally, Datum provides $> 45\%$ improvement over current design proposals for geo-distributed data analytics systems. Datum works by decomposing the total cost minimization problem into subproblems that allow optimization of data purchasing and data placement separately, which provides a practical route for implementation in real systems. Further, Datum provides a unified solution across systems using per-query pricing or bulk pricing, systems with data replication constraints and/or regulatory constraints on data placement, and systems with SLA constraints on delivery.

BIBLIOGRAPHY

[1] Khalid Al-Sultan and M. Al-Fawzan. "A Tabu Search Approach to the Uncapacitated Facility Location Problem". In: *Annals of Operations Research* (1999).

[2] Réka Albert, Hawoong Jeong, and Albert-László Barabási. "Internet: Diameter of the world-wide web". In: *Nature* 401.6749 (1999), pp. 130–131.

[3] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. "Space-Efficient Local Computation Algorithms". In: *Proc./ 22ndACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2012, pp. 1132–1139.

[4] Reid Andersen et al. "Local Computation of PageRank Contributions". In: *Internet Mathematics* 5(1–2) (2008), pp. 23–45.

[5] Lachlan LH Andrew et al. "A Tale of Two Metrics: Simultaneous Bounds on Competitiveness and Regret." In: *COLT*. 2013, pp. 741–763.

[6] Magdalena Balazinska, Bill Howe, and Dan Suciu. "Data Markets in the Cloud: An Opportunity for the Database Community". In: *Proceedings of the VLDB Endowment* (2011).

[7] Magdalena Balazinska et al. "A Discussion on Pricing Relational Data". In: *In Search of Elegance in the Theory and Practice of Computation*. 2013.

[8] John Beasley. "Lagrangean Heuristics for Location Problems". In: *European Journal of Operational Research* (1993).

[9] Jacques F Benders. "Partitioning procedures for solving mixed-variables programming problems". In: *Numerische mathematik* 4.1 (1962), pp. 238–252.

[10] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[11] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. 1997.

[12] Vincent D Blondel, Julien M Hendrickx, Alex Olshevsky, and John N Tsitsiklis. "Convergence in multiagent coordination, consensus, and flocking". In: *Proceedings of IEEE Conference on Decision and Control*. IEEE. 2005, pp. 2996–3000.

[13] Sem Borst, Varun Gupta, and Anwar Walid. "Distributed caching algorithms for content distribution networks". In: *Proceedings of IEEE INFOCOM*. IEEE. 2010, pp. 1–9.

[14] Stephen Boyd et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends® in Machine Learning* 3.1 (2011), pp. 1–122.

[15] Niv Buchbinder and Joseph Naor. "The Design of Competitive Online Algorithms via a Primal-Dual Approach". In: *Foundations and Trends in Theoretical Computer Science* 3.2-3 (2009), pp. 93–263.

[16] Yijia Cao et al. "An optimized EV charging model considering TOU price and SOC curve". In: *IEEE Transactions on Smart Grid* 3.1 (2012), pp. 388–393.

[17] Moses Charikar, Sudipto Guha, Éva Tardos, and David Shmoys. "A Constant-factor Approximation Algorithm for the K-median Problem (Extended Abstract)". In: *STOC*. 1999.

[18] Mung Chiang, Steven H Low, A Robert Calderbank, and John C Doyle. "Layering as optimization decomposition: A mathematical theory of network architectures". In: *Proceedings of the IEEE* 95.1 (2007), pp. 255–312.

[19] Patrick L Combettes and Jean-Christophe Pesquet. "A Douglas–Rachford splitting approach to nonsmooth convex variational signal recovery". In: *IEEE Journal of Selected Topics in Signal Processing* 1.4 (2007), pp. 564–574.

[20] Patrick L Combettes and Valérie R Wajs. "Signal recovery by proximal forward-backward splitting". In: *Multiscale Modeling & Simulation* 4.4 (2005), pp. 1168–1200.

[21] James Corbett et al. "Spanner: Google's Globally Distributed Database". In: *ACM Transactions on Computer Systems* (2013).

[22] Rachel Cummings et al. "Accuracy for Sale: Aggregating Data with a Variance Constraint". In: *ITCS*. 2015.

[23] George Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

[24] George B Dantzig and Philip Wolfe. "Decomposition principle for linear programs". In: *Operations research* 8.1 (1960), pp. 101–111.

[25] Cynthia Dwork. "Differential Privacy". In: *Encyclopedia of Cryptography and Security*. 2011.

[26] Donald Erlenkotter. "A Dual-Based Procedure for Uncapacitated Facility Location". In: *Operations Research* (1978).

[27] Tomaso Erseghe. "Distributed optimal power flow using ADMM". In: *IEEE transactions on power systems* 29.5 (2014), pp. 2370–2380.

[28] Hugh Everett III. "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources". In: *Operations research* 11.3 (1963), pp. 399–417.

[29] *Factual*. `https://www.factual.com/`. 2015.

[30] Uriel Feige. "A Threshold of ln n for Approximating Set Cover". In: *J. ACM* (1998).

[31] Uriel Feige, Boaz Patt-Shamir, and Shai Vardi. *On the probe complexity of local computation algorithms*. Under submission. 2017.

[32] Lisa Fleischer and Yu han Lyu. "Approximately Optimal Auctions for Selling Privacy when Costs are Correlated with Data". In: *Proceedings of the 13th ACM Conference on Electronic Commerce*. 2012.

[33] Pedro A Forero, Alfonso Cano, and Georgios B Giannakis. "Consensus-based distributed support vector machines". In: *Journal of Machine Learning Research* 11.May (2010), pp. 1663–1707.

[34] Daniel Gabay and Bertrand Mercier. "A dual algorithm for the solution of nonlinear variational problems via finite element approximation". In: *Computers & Mathematics with Applications* 2.1 (1976), pp. 17–40.

[35] Lingwen Gan, Ufuk Topcu, and Steven Low. "Optimal decentralized protocol for electric vehicle charging". In: *IEEE Transactions on Power Systems* 28.2 (2013), pp. 940–951. ISSN: 0885-8950. DOI: `10.1109/TPWRS.2012.2210288`.

[36] Diptesh Ghosh. "Neighborhood Search Heuristics for the Uncapacitated Facility Location Problem ". In: *European Journal of Operational Research* (2003).

[37] *Google Data Center FAQ*. `http://www.datacenterknowledge.com/archives/2012/05/15/google-data-center-faq/`. 2012.

[38] Sudipto Guha and Samir Khuller. "Greedy Strikes Back: Improved Facility Location Algorithms". In: *Journal of Algorithms* (1999).

[39] Yi Guo and Lynne E Parker. "A distributed and optimal motion planning approach for multiple mobile robots". In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 3. IEEE. 2002, pp. 2612–2619.

[40] Ashish Gupta et al. "Mesa: Geo-replicated, Near Real-time, Scalable Data Warehousing". In: *Proceedings of the VLDB Endowment* (2014).

[41] Elad Hazan. "Introduction to online convex optimization". In: *Foundations and Trends® in Optimization* 2.3-4 (2016), pp. 157–325.

[42] Dorit Hochbaum. "Heuristics for the Fixed Cost Median Problem". In: *Math. Program.* (1982).

[43] Chien-Chun Hung, Leana Golubchik, and Minlan Yu. "Scheduling Jobs across Geo-distributed Datacenters". In: *Proceedings of the 6th ACM Symposium on Cloud Computing*. 2015.

[44] *Infochimps*. `http://www.infochimps.com/`. 2015.

[45] Kamal Jain and Vijay Vazirani. "Approximation Algorithms for Metric Facility Location and k-Median Problems Using the Primal-dual Schema and Lagrangian Relaxation". In: *J. ACM* (2001).

[46] Jonathan Katz and Luca Trevisan. "On the efficiency of local decoding procedures for error-correcting codes". In: *Proc. 32nd Annual ACM Symposium on the Theory of Computing (STOC)*. 2000, pp. 80–86.

[47] Frank P Kelly, Aman K Maulloo, and David KH Tan. "Rate control for communication networks: shadow prices, proportional fairness and stability". In: *Journal of the Operational Research society* 49.3 (1998), pp. 237–252.

[48] Manfred Korkel. "On the Exact Solution of Large-scale Simple Plant Location Problems ". In: *European Journal of Operational Research* (1989).

[49] Paraschos Koutris et al. "Query-based Data Pricing". In: *Proceedings of the 31st symposium on Principles of Database Systems*. 2012.

[50] Paraschos Koutris et al. "QueryMarket Demonstration: Pricing for Online Data Markets". In: *Proceedings of the VLDB Endowment* (2012).

[51] Paraschos Koutris et al. "Toward Practical Query Pricing with QueryMarket". In: *SIGMOD*. 2013.

[52] Jakob Krarup and Peter Pruzan. "The Simple Plant Location Problem: Survey and Synthesis". In: *European Journal of Operational Research* (1983).

[53] Yoshiaki Kuwata and Jonathan P How. "Cooperative distributed robust trajectory optimization using receding horizon MILP". In: *IEEE Transactions on Control Systems Technology* 19.2 (2011), pp. 423–431.

[54] Leon S Lasdon. *Optimization theory for large systems*. Courier Corporation, 1970.

[55] George Lee et al. "The Unified Logging Infrastructure for Data Analytics at Twitter". In: *Proceedings of the VLDB Endowment* (2012).

[56] Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. "Brief Announcement: Local Computation Algorithms for Graphs of Non-Constant Degrees". In: *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA*. 2015, pp. 59–61.

[57] Chao Li, Daniel Yang Li, Gerome Miklau, and Dan Suciu. "A Theory of Pricing Private Data". In: *ACM Transactions on Database Systems* (2014).

[58] Na Li, Lijun Chen, and Steven H Low. "Optimal demand response based on utility maximization in power networks". In: *Power and Energy Society General Meeting, 2011 IEEE*. IEEE. 2011, pp. 1–8.

[59] Ying Liao, Huan Qi, and Weiqun Li. "Load-balanced clustering algorithm with distributed self-organization for wireless sensor networks". In: *IEEE Sensors Journal* 13.5 (2013), pp. 1498–1506.

[60] Palma London, Niangjun Chen, Shai Vardi, and Adam Wierman. *Distributed Optimization via Local Computation Algorithms*. `http://users.cms.caltech.edu/~plondon/loco.pdf`. Under submission. 2017.

[61] Steven Low and David Lapsley. "Optimization flow control. I. Basic algorithm and convergence". In: *IEEE/ACM Transactions on Networking* 7.6 (1999), pp. 861–874. ISSN: 1063-6692. DOI: 10.1109/90.811451.

[62] Steven H Low, Fernando Paganini, and John C Doyle. "Internet congestion control". In: *IEEE control systems* 22.1 (2002), pp. 28–43.

[63] Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. "Converting Online Algorithms to Local Computation Algorithms". In: *Proceedings of 39th International Colloquium on Automata, Languages and Programming (ICALP)*. 2012, pp. 653–664.

[64] Laurent Massoulié and James Roberts. "Bandwidth sharing: objectives and algorithms". In: *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 3. IEEE. 1999, pp. 1395–1403.

[65] *Microsoft Azure*. https://azure.microsoft.com/en-us/. 2015.

[66] Angelia Nedić and Asuman Ozdaglar. "Convergence rate for consensus with delays". In: *Journal of Global Optimization* 47.3 (2010), pp. 437–456.

[67] Angelia Nedic and Asuman Ozdaglar. "Distributed subgradient methods for multi-agent optimization". In: *IEEE Transactions on Automatic Control* 54.1 (2009), pp. 48–61.

[68] M. E. J. Newman. "Power Laws, Pareto Distributions and Zipf's Law". In: *Contemporary physics* (2005).

[69] Reza Olfati-Saber. "Distributed Kalman filtering for sensor networks". In: *Decision and Control, 2007 46th IEEE Conference on*. IEEE. 2007, pp. 5492–5498.

[70] Venkata N Padmanabhan, Helen J Wang, Philip A Chou, and Kunwadee Sripanidkulchai. "Distributing streaming media content using cooperative networking". In: *Proceedings of workshop on Network and operating systems support for digital audio and video*. ACM. 2002, pp. 177–186.

[71] Daniel P Palomar and Mung Chiang. "Alternative distributed algorithms for network utility maximization: Framework and applications". In: *IEEE Transactions on Automatic Control* 52.12 (2007), pp. 2254–2269.

[72] Qiuyu Peng and Steven Low. "Distributed optimal power flow algorithm for radial networks, I: Balanced single phase case". In: *IEEE Transactions on Smart Grid* (2016).

[73] Qifan Pu et al. "Low Latency Geo-distributed Data Analytics". In: *SIGCOMM*. 2015.

[74] A. Rabkin et al. "Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area". In: *NSDI*. 2014.

[75] Robin L Raffard, Claire J Tomlin, and Stephen P Boyd. "Distributed optimization for cooperative agents: Application to formation flight". In: *Decision and Control, 2004. CDC. 43rd IEEE Conference on*. Vol. 3. IEEE. 2004, pp. 2453–2459.

[76] Omer Reingold and Shai Vardi. "New techniques and tighter bounds for local computation algorithms". In: *Journal of Computer and System Science* 82.7 (2016), pp. 1180–1200.

[77] Xiaoqi Ren, Palma London, Juba Ziani, and Adam Wierman. *Joint Data Purchasing and Data Placement in a Geo-Distributed Data Market*. Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science. 2016.

[78] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. "Fast Local Computation Algorithms". In: *Proc. 2nd Symposium on Innovations in Computer Science (ICS)*. 2011, pp. 223–238.

[79] Michael E. Saks and C. Seshadhri. "Local Monotonicity Reconstruction". In: *SIAM Journal on Computing* 39.7 (2010), pp. 2897–2926.

[80] Pedram Samadi et al. "Optimal real-time pricing algorithm based on utility maximization for smart grid". In: *Proceedings of Smart Grid Communications (SmartGridComm)*. IEEE. 2010, pp. 415–420.

[81] Ioannis D Schizas, Alejandro Ribeiro, and Georgios B Giannakis. "Consensus in ad hoc WSNs with noisy linksPart I: Distributed estimation of deterministic signals". In: *IEEE Trans. on Signal Processing* 56.1 (2008), pp. 350–364.

[82] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[83] Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*. Vol. 3. Springer Science & Business Media, 2012.

[84] Rayadurgam Srikant. *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012.

[85] Gabriele Steidl and Tanja Teuber. "Removing multiplicative noise by Douglas-Rachford splitting methods". In: *Journal of Mathematical Imaging and Vision* 36.2 (2010), pp. 168–184.

[86] Ichiro Suzuki and Masafumi Yamashita. "Distributed anonymous mobile robots: Formation of geometric patterns". In: *SIAM Journal on Computing* 28.4 (1999), pp. 1347–1363.

[87] *The CAIDA UCSD AS Relationship Dataset, September 17, 2007*. `http://www.caida.org/data/as-relationships/`.

[88] *The IUPHAR/BPS Guide to Pharmacology*. `http://www.guidetopharmacology.org/`. 2015.

[89] Dilek Tuzun and Laura Burke. "A Two-phase Tabu Search Approach to the Location Routing Problem ". In: *European Journal of Operational Research* (1999).

[90] Vijay Vazirani. *Approximation Algorithms*. Springer, 2001.

[91] *Visipedia Project*. `http://www.vision.caltech.edu/visipedia/`. 2015.

[92] Ashish Vulimiri et al. "Global Analytics in the Face of Bandwidth and Regulatory Constraints". In: *NSDI*. 2015.

[93] Ashish Vulimiri et al. "WANalytics: Analytics for a Geo-distributed Data-intensive World". In: *CIDR*. 2015.

[94] Janet Wiener and Nathan Boston. *Facebook's top open data problems*. `https://research.facebook.com/blog/1522692927972019/facebook-s-top-open-data-problems/`. 2014.

[95] *Xignite*. `http://www.xignite.com/`. 2015.

[96] Yung Yi and Mung Chiang. "Stochastic network utility maximization - a tribute to Kelly's paper published in this journal a decade ago". In: *European Transactions on Telecommunications* 19.4 (2008), pp. 421–442.

[97] Jiawei Zhang. "Approximating the two-level facility location problem via a quasi-greedy approach". In: *Mathematical Programming* (2006).

A p p e n d i x A

PSEUDOCODE FOR GENERAL ONLINE FRACTIONAL PACKING

The following pseudocode is replicated from [15]. Constraints arrive in some order. During the jth round, the dual variable $y(j)$ and all the primal variables are increased. The minimum $y(j)$ is found, such that the primal constraints are satisfied.

Algorithm 1: General Online Fractional Packing

Input: $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$
Output: x, y
Initialize $x = \mathbf{0_n}$, $y = \mathbf{0_m}$ **for** $j = 1...m$ **do**
  for $i = 1...n$ **do**
    $a_i(\max) \leftarrow \max_{k=1}^{j}\{a(i,k)\}$
  while $\sum_{i=1}^{n} a(i,j)x(i) < 1$ **do**
    Increase $y(j)$ continuously
    for $i = 1...n$ **do**
      $\delta = \exp(\frac{B}{2c(i)} \sum_{k=1}^{j} a(i,k)y(k)) - 1$
      $x(i) = \max\left\{x(i), \frac{1}{na_i(\max)}\delta\right\}$

Instead of increasing $y(j)$ continuously, one can perform a binary search over possible values of $y(j)$. For each candidate $y(j)$, a corresponding new value of x is computed and the primal constraints are checked for feasibility. If feasible, the new x is accepted, and $y(j)$ will be increased in the next round of the binary search. If infeasible, the new x is rejected, and the value of $y(j)$ will be decreased in the next round of the search.

A.1 ADMM

In our numerical results we compare LOCO to ADMM in the case of linear NUM. For completeness, we describe the application of ADMM to that setting here.

To apply ADMM, we first absorb the inequality constraint $x \leq \bar{x}$ into the inequality $A''x \leq c'$ by letting $A'' = \left[A, I\right]^T$ and $c' = \left[c, \bar{x}\right]^T$, where this

notation indicates a stack of vectors. We introduce a slack variable $s \geq 0$ such that the inequality constraint becomes $A''x + s = c'$. Let $x' = \Big[x, s\Big]^T$, $A' = [A'' \quad I]$ and $b = \Big[\mathbf{1}_n, \mathbf{0}_n\Big]^T$. We can now write the problem in standard ADMM form,

$$\begin{aligned} \min_{x',z} \quad & g(x') + h(z) \\ \text{s.t.} \quad & x' - z = 0 \end{aligned}$$

where $g = (x - \underline{x})_+$ is the indicator function associated with the constraints $\underline{x} \leq x$ and $h(z') = -b^T z$ where dom $h = \{z | A'z = c'\}$.

Writing down the scaled augmented Lagrangian $L_\rho(x', z, u) = g(x') + h(z) + u^T(z - x') + \frac{\rho}{2}\|x' - z\|^2$, we can see that all the update steps have closed form solution (see [14, Chapter 5.2]). The updates become:

$$\begin{aligned} x'^{k+1} &= (z^{k+1} + u^k - \underline{x})_+ \quad \forall i \\ z^{k+1} &= \begin{bmatrix} \rho I & A'^T \\ A' & 0 \end{bmatrix}^{-1} \begin{bmatrix} \rho(x'^k - u^k) - b \\ c' \end{bmatrix} \\ u^{k+1} &= u^k + (x'^{k+1} - z^{k+1}) \quad \forall i \end{aligned}$$

The solution to the NUM problem is recovered from the first n entries of x'.

A p p e n d i x B

PROOF OF LEMMA 3

We denote the set $\{0, 1, \ldots, m\}$ by $[m]$. Logarithms are base e. Let $G = (V, E)$ be a graph. For any vertex set $S \subseteq V$, denote by $N(S)$ the set of vertices that are not in S but are neighbors of some vertex in S: $N(S) = \{N(v) : v \in S\} \setminus S$. The *length* of a path is the number of edges it contains. For a set $S \subseteq V$ and a function $f : V \to \mathbb{N}$, we use $S \cap f^{-1}(i)$ to denote the set $\{v \in S : f(v) = i\}$.

Let $G = (V, E)$ be a graph, and let $f : V \to \mathbb{N}$ be some function on the vertices. An *adaptive vertex exposure procedure* A is one that does not know f a priori. A is given a vertex $v \in V$ and $f(v)$; A iteratively adds vertices from $V \setminus S$ to S: for every vertex u that A adds to S, $f(u)$ is revealed immediately after u is added. Let S^t denote S after the addition of the t^{th} vertex. The following is a simple concentration bound whose proof is given for completeness.

Lemma 10 *Let $G = (V, E)$ be a graph, let $Q > 0$ be some constant, let $\gamma = 15Q$, and let $f : V \to [Q]$ be a function chosen uniformly at random from all such possible functions. Let A be an adaptive vertex exposure procedure that is given a vertex $v \in V$. Then, for any $q \in [Q]$, the probability that there is some t, $\gamma \log n \leq t \leq n$ for which $|S^t \cap f^{-1}(q)| > \frac{2|S^t|}{Q}$ is at most $\frac{1}{n^4}$.*

Proof 11 *Let v_j be the j^{th} vertex added to S by A, and let X_j be the indicator variable whose value is 1 iff $f(v_j) = q$. For any $t \leq n$, $\mathbb{E}\left[\sum_{j=1}^{t} X_j\right] = \frac{t}{Q}$. As X_i and X_j are independent for all $i \neq j$, by the Chernoff bound, for $\gamma \log n \leq t \leq n$,*

$$\Pr\left[\sum_{j=1}^{t} X_j > \frac{2t}{Q}\right] \leq e^{\frac{-t}{3Q}} \leq e^{-5 \log n}.$$

A union bound over all possible values of $t : \gamma \log n \leq t \leq n$ completes the proof.

Let $r : V \to [0, 1]$ be a function chosen uniformly at random from all such possible functions. Partition $[0, 1]$ into $Q = 4(d+1)$ segments of equal measure,

$I_1, \ldots, I_Q$. For every $v \in V$, set $f(v) = q$ if $r(v) \in I_q$ (f is a quantization of r).

Consider the following method of generating two sets of vertices: T and R, where $T \subseteq R$. For some vertex v, set $T = R = \{v\}$. Continue inductively: choose some vertex $w \in T$, add all $N(w)$ to R and compute $f(u)$ for all $u \in N(w)$. Add the vertices u such that $u \in N(w)$ and $f(u) \geq f(w)$ to T. The process ends when no more vertices can be added to T. T is the query set with respect to f, hence $|T|$ is an upper bound on the size of the actual query set (i.e., the query set with respect to r). However, it is difficult to reason about the size of T directly, as the ranks of its vertices are not independent. The ranks of the vertices in R, though, *are* independent, as R is generated by an adaptive vertex exposure procedure. R is a superset of T that includes T and its boundary, hence $|R|$ is also an upper bound on the size of the query set.

We now define $Q+1$ "layers" - $T_{\leq 0}, \ldots, T_{\leq Q}$: $T_{\leq q} = T \cap \bigcup_{i=0}^{q} f^{-1}(i)$. That is, $T_{\leq q}$ is the set of vertices in T whose rank is at most q. (The range of f is $[Q]$, hence $T_{\leq 0}$ will be empty, but we include it to simplify the proof.)

Claim 12 *Set $Q = 4(d+1)$, $\gamma = 15Q$. Assume without loss of generality that $f(v) = 0$. Then for all $0 \leq i \leq Q-1$,*

$$\Pr[|T_{\leq i}| \leq 2^i \gamma \log n \wedge |T_{\leq i+1}| \geq 2^{i+1} \gamma \log n] \leq \frac{1}{n^4}.$$

Proof 13 *For all $0 \leq i \leq Q$, let $R_{\leq i} = T_{\leq i} \cup N(T_{\leq i})$. Note that*

$$R_{\leq i} \cap f^{-1}(i) = T_{\leq i} \cap f^{-1}(i), \tag{B.1}$$

because if there had been some $u \in N(T_{\leq i}), f(u) = i$, u would have been added to $T_{\leq i}$.

Note that $|T_{\leq i}| \leq 2^i \gamma \log n \wedge |T_{\leq i+1}| \geq 2^{i+1} \gamma \log n$ implies that

$$|T_{\leq i+1} \cap f^{-1}(i+1)| > \frac{|T_{\leq i+1}|}{2}. \tag{B.2}$$

In other words, the majority of vertices $v \in T_{\leq i+1}$ must have $f(v) = i+1$.

Given $|T_{\leq i+1}| > 2^{i+1} \gamma \log n$, it holds that $|R_{\leq i+1}| > 2^{i+1} \gamma \log n$ because $T_{\leq i+1} \subseteq R_{\leq i+1}$. Furthermore, $R_{\leq i+1}$ was constructed by an adaptive vertex exposure

procedure and so the conditions of Lemma 10 hold for $R_{\leq i+1}$. *From Equations* (B.1) *and* (B.2) *we get*

$$\begin{aligned}
&\Pr[|T_{\leq i}| \leq 2^i \gamma \log n \wedge |T_{\leq i+1}| \geq 2^{i+1}\gamma \log n] \\
&\leq \Pr\left[\left|R_{\leq i+1} \cap f^{-1}(i+1)\right| > \frac{|T_{\leq i+1}|}{2}\right] \\
&\leq \Pr\left[\left|R_{\leq i+1} \cap f^{-1}(i+1)\right| > \frac{2\,|R_{\leq i+1}|}{Q}\right] \\
&\leq \frac{1}{n^4},
\end{aligned}$$

where the second inequality is because $|R_{\leq i+1}| \leq (d+1)|T_{\leq i+1}|$, *as* G*'s degree is at most* d*; the last inequality is due to Lemma 10.*

Lemma 14 *Set* $Q = 4(d+1)$. *Let* $G = (V, E)$ *be a graph with degree bounded by* d, *where* $|V| = n$. *For any vertex* $v \in G$, $\Pr\left[T_v > 2^Q \cdot 15Q \log n\right] < \frac{1}{n^3}$.

Proof 15 *To prove Lemma 14, we need to show that, for* $\gamma = 15Q$,

$$\Pr[|T_{\leq Q}| > 2^L \gamma \log n] < \frac{1}{n^3}.$$

We show that for $0 \leq i \leq Q, \Pr[|T_{\leq i}| > 2^i \gamma \log n] < \frac{i}{n^4}$, *by induction. For the base of the induction,* $|S_0| = 1$, *and the claim holds. For the inductive step, assume that* $\Pr[|T_{\leq i}| > 2^i \gamma \log n] < \frac{i}{n^4}$. *Then, denoting by* X *the event* $|T_{\leq i}| > 2^i \gamma \log n$ *and by* $\bar{X}$ *the event* $|T_{\leq i}| \leq 2^i \gamma \log n$,

$$\begin{aligned}
&\Pr[|T_{\leq i+1}| > 2^{i+1}\gamma \log n] \\
&= \Pr[|T_{\leq i+1}| > 2^{i+1}\gamma \log n : X] \Pr[X] \\
&+ \Pr[|T_{\leq i+1}| > 2^{i+1}\gamma \log n : \bar{X}] \Pr[\bar{X}].
\end{aligned}$$

From the inductive step and Claim 12, using the union bound, the lemma follows.

Applying a union bound over all the vertices gives the size of *each* query set is $O(\log n)$ with probability at least $1 - 1/n^2$, completing the proof of Theorem 3.

A p p e n d i x C

PROOF OF THEOREM 8

To prove Theorem 8, we show a connection between the data cloud cost minimization problem in (9.5) and the uncapacitated facility location problem. In particular, we show both that the facility location problem can be reduced to a data cloud optimization problem and vice versa.

First, we show that every instance of the uncapacitated facility location problem can be viewed as an instance of (9.5).

Take any instance of the uncapacitated facility location problem (UFLP). Let I be the set of customers, J the set of locations, α_{ij} the cost of assigning customer i to location j, and β_j the cost of opening facility j. Binary variables $y_j = 1$ if and only if facility is open at site j, and $x_{j,i} = 1$ if and only if customer i is assigned to location j. Then the UFLP can be formulated as following.

$$\min_{x,y} \quad \sum_{j \in F} \beta_j y_j + \sum_{i \in I, j \in F} \alpha_{ij} x_{j,i} \tag{C.1}$$

subject to

$$x_{j,i} \leq y_j, \ \forall i, j$$

$$\sum_{j \in F} x_{j,i} = 1, \ \forall c$$

$$x_{j,i}, y_j \in \{0, 1\}, \forall i, j$$

Mapping j to d and i to c yields an instance of (9.5) with $|P| = |L| = 1$, $f(l) = 0$ and $w_c(l) = 0$, in which case constraint (9.5c) becomes trivial.

Next, we show that every instance of (9.5) can be written as an instance of UFLP.

We start by remarking that (9.5) (with p dropped) is equivalent to the following

ILP.

$$\min_{x,y} \quad \sum_{d,l=1}^{D,L} \beta_d(l) y_d(l) + \sum_{d,l,c=1}^{D,L,C} (f(l) + \alpha_{d,c}(l)) x_{d,c}(l) \tag{C.2}$$

subject to

$$x_{d,c}(l) \leq y_d(l),\ \forall c, l, d$$
$$\sum_{d=1}^{D} \sum_{l=1}^{L} x_{d,c}(l) = 1,\ \forall c$$
$$x_{d,c}(l), y_d(l) \in \{0,1\}, \forall c, l, d$$

with $\alpha_{d,c}(l) = M$, for M big enough, whenever $l < w_c$. Indeed, in any feasible solution of (9.5), we necessarily have $x_{d,c}(l) = 0$ whenever $l < w_c$, as each client purchases exactly one quality level and this quality level has to be higher than the minimum required level w_c; by setting $\alpha_{d,c}(l)$ big enough, we ensure that any optimal solution must have $x_{d,c}(l) = 0$ thus must be feasible for (9.5), and has the same cost as in (9.5). Now, take $J = [D] \times [L]$ and $I = [C]$, and the problem can be rewritten as

$$\min_{x,y} \quad \sum_{(d,l)\in J} \beta_d(l) y_d(l) + \sum_{(d,l)\in J, c\in I} (f(l) + \alpha_{d,c}(l)) x_{d,c}(l) \tag{C.3}$$

$$\text{subject to} \quad x_{d,c}(l) \leq y_d(l),\ \forall (d,l) \in J, c \in I$$
$$\sum_{(d,l)\in J} x_{d,c}(l) = 1,\ \forall c \in I$$
$$x_{d,c}(l), y_d(l) \in \{0,1\}, \forall c \in I, (d,l) \in J$$

which is an UFLP.

C.1 Proof of Theorem 9

Assume without loss of generality that all clients can be satisfied by the highest quality level, i.e., $w_c \leq q(L), \forall c$. Define $C_i = \{c : q(i-1) < w_c \leq q(i)\}$ (q(0) = 0 by default). Given these assumptions, clients can be grouped into L categories $\{C_1, C_2, \ldots, C_L\}$ based on their minimum quality level. Note that $C_i \cap C_j = \emptyset, \forall i, j$ and $\cup_{i=1}^{L} C_i = C$. Without loss of generality, assume $C_i \neq \emptyset,\ \forall i$.

As the clients in the same group C_i all face exactly the same choice of quality levels and minimum quality requirements, there must always be an optimal solution in which the data purchasing decisions of any clients within one category are the same.

Let us denote the number of clients in category C_i by S_i. Denote the purchasing decision of category C_i by χ_i, e.g., $\chi_i(l) = x_c(l),\ \forall l, c \in C_i$, similar to the argument in proof of Theorem 8, we can reformulate (10.2) as follows. Note the slight abuse of notation, as clients and their associated required quality level are represented by the same letter, i, due to clients in category C_i having minimum quality level i by definition.

$$\text{minimize} \quad \sum_{l=1}^{L} \beta(l)y(l) + \sum_{i=1}^{L}\sum_{l=i}^{L} S_i f(l)\chi_i(l) \tag{C.4}$$

$$\text{subject to} \quad \chi_i(l) \le y(l),\ \forall i, l \tag{C.4a}$$

$$\sum_{l=i}^{L} \chi_i(l) = 1,\ \forall i \tag{C.4b}$$

$$\chi_i(l) \ge 0, \forall i, l \tag{C.4c}$$

$$y(l) \ge 0,\ \forall l \tag{C.4d}$$

$$\chi_i(l), y(l) \in \{0, 1\}, \forall i, l \tag{C.4e}$$

Consider the linear relaxation of (C.4), which drops the $0-1$ integer constraint (C.4e). For any optimal solution $\{\chi_i^r(l), y^r(l)\}$ of the linear relaxation we have the following observations.

1. $\chi_L^r(L) = 1$.

   Proof 16 *From* (C.4b)*, let* $i = L$*, then* $\chi_L^r(L) = 1$*. The intuition behind this is that, since* $C_L \neq \emptyset$*, highest quality data always has to be purchased to provide service for clients in* $C(L)$*.*

2. $y^r(l) = \max_i\{\chi_i^r(l)\} \in [0, 1]$ and $y^r(L) = 1$.

   Proof 17 *From* (C.4a)*, the non-negativity of* $\{\beta(l)\}$*, and the optimality of* $\{y^r(l)\}$*,* $y^r(l) = \max_i\{\chi_i^r(l)\}$*. From the non-negativity of* $\{\chi_i^r(l)\}$*,* $y^r(l) = \max_i\{\chi_i^r(l)\} \le \sum_{l=i}^{L} \chi_i^r(l) = 1$*, and* $y^r(L) = \chi_L^r(L) = 1$

3. $\forall l \ge i$, if $\sum_{l=i}^{L} y^r(l) \le 1, \chi_i^r(l) = y^r(l)$; otherwise, $\chi_i(l) = \max\{1 - \sum_{k=i}^{l-1} y^r(k), 0\}$.

   Proof 18 *For some fixed* i*,* $\{S_i f(l)\}$ *is a positive, strictly increasing sequence as* l *increases. From constraint* (C.4a) *and* (C.4b)*,* $\chi_i^r(l) \le y^r(l)$*,*

and $\sum_{l=i}^{L} \chi_i^r(l) = 1$. *Since* $\{\chi_i^r(l), y^r(l)\}$ *is optimal,* $\forall l \geq i$, *if* $\sum_{k=i}^{l} y^r(k) \leq 1, \chi_i^r(l) = y^r(l)$; *otherwise,* $\chi_i^r(l) = \max\{1 - \sum_{k=i}^{l-1} y^r(k), 0\}$.

Next, define $m_i \in \{i, \ldots, n\}$ such that $\sum_{l=i}^{m_i-1} y^r(l) < 1$, and $\sum_{l=i}^{m_i} y^r(l) \geq 1$. Such an m_i must exist since $y^r(l) \geq 0$ for all l and $y^r(L) = 1$. Recall $\chi_L^r(L) = y^r(L) = 1$. For for any $i = 1, 2, \ldots, L-1$, if the values of $\{y^r(l)\}$ are given, the optimal $\{\chi_i^r(l)\}$ satisfy the following closed form expression:

$$\chi_i^r(l) = \begin{cases} y^r(l), & i \leq l < m_i \\ 1 - \sum_{k=i}^{m_i-1} y^r(k), & l = m_i \\ 0, & m_i < l \leq n. \end{cases} \tag{C.5}$$

Note that, if y^r are binary, then χ^r are binary. Suppose there exists an optimal solution $\{\chi^r, y^r\}$ with $y^r \notin \{0,1\}^L$, in the following we show that there exists a feasible binary solution $\{\chi^*, y^*\}$ of (C.4) such that the objective value generated by $\{\chi^*, y^*\}$ is better than or equal to that of $\{\chi^r, y^r\}$.

Suppose fractional solution y^r is an optimal solution of the linear relaxation and calculate m_i as in (C.5). Write χ as a function of y, $\forall i, l$.

$$\chi_i(l) = \begin{cases} y(l), & i \leq l < m_i \\ 1 - \sum_{k=i}^{m_i-1} y(k), & l = m_i \\ 0, & m_i < l \leq n \end{cases} \tag{C.6}$$

Substituting (C.6) in the objective function (C.4), the objective function becomes a linear combination of $\{y(l)\}$ that we denote $L(y)$.

Consider the optimization problem in which $\{\chi_i(l)\}$ is expressed as a function of $\{y(l)\}$ in the linear relaxation:

$$\begin{aligned} \text{minimize} \quad & L(y) \\ \text{subject to} \quad & \sum_{l=i}^{m_i'-1} y(l) \leq 1, \ \forall i = 1, \ldots, L-1 \\ & \sum_{l=i}^{m_i'} y(l) \geq 1, \ \forall i = 1, \ldots, L-1 \\ & y(l) \geq 0, \ \forall l = 1, \ldots, L \\ & y(L) = 1 \end{aligned} \tag{C.7}$$

The following claims hold:

1. (C.7) is feasible and bounded, and always has an optimal solution at an extreme point.

   Proof 19 *Clearly,* $\forall l, y(l) \in [0,1]$. *And starting from* $y(L)$, *it is easy to construct a feasible solution of* (C.7). *Thus,* (C.7) *is feasible and bounded, and always has an optimal solution at an extreme point.*

2. $\{y^r(l)\}$ is a feasible solution of (C.7).

   Proof 20 *Since* $\{y^r(l)\}$ *is feasible for* (C.4), $y^r(l) \geq 0$, $\forall l$, *and* $y^r(L) = 1$. *By definition of* m_i, $\sum_{l=i}^{m_i-1} y^r(l) \leq 1$, $\sum_{l=i}^{m_i} y^r(l) \geq 1$.

3. Any extreme point $\{y(l)\}$ of (C.7) is binary.

   Proof 21 *Since* $y(L) = 1$, *we can drop* $y(L)$, *and write* (C.7) *in the following standard linear programming form:*

$$\begin{aligned} \min_{y} \quad & L(y) \\ s.t. \quad & Ay \leq b \\ & y \geq 0 \end{aligned} \tag{C.8}$$

   Note that all entries of A *are* $0, \pm 1$, *and all rows of* A *has either consecutive* 1*s or consecutive* -1*s. Thus, from [82],* A *is a totally unimodular matrix thus the extreme points of* (C.8) *are all integral. In particular, since all* $y(l) \in [0,1]$, *the extreme points of* (C.8) *are all binary.*

4. The $\{\chi_i^*(l)\}$ obtained through (C.6) corresponding to an optimal binary solution $\{y^*\}$ is also binary.

   Proof 22 *Follows immediately from* (C.6) *and integrality of* $\{y^*(l)\}$.

5. $\{\chi_i^*(l), y^*(l)\}$ is a feasible solution of the linear relaxation of (C.4).

   Proof 23 *Follows from* (C.6) *and* $\sum_{l=i}^{L} \chi_i^*(l) = 1$

   $\{\chi_i^r(l), y^r(l)\}$ and any optimal extreme point $\{\chi_i^*(l), y^*(l)\}$ see their corresponding objective values unchanged between (C.7) and the relaxation of (C.4) by construction of the $\chi_i(l)$'s. And any such extremal and optimal

$\{\chi_i^*(l), y^*(l)\}$ has a better or equal objective value compared to $\{\chi_i^r(l), y^r(l)\}$ in relaxed (C.4). Since $\{\chi_i^r(l), y^r(l)\}$ is optimal for (C.7), it implies any optimal extreme point of relaxed (C.4) yields a binary and optimal solution for (C.7). This provides a polynomial time algorithm to find such a binary optimal solution, which can be summarized as in §10.2.

C.2 Proof of Step 2 in §10.2

In this section we derive the closed form solutions of (10.7) for the optimization in (10.6). We start by discussing the form of $x_{v,c}(l)$. Consider the following two cases based on the value of $Y(l)$.

1. For any quality level l', if $Y(l') = 0$, then $\forall v, \sum_{v=1}^{V} y_v(l') = Y(l') = 0$. From the non-negativity of $y_v(l')$, $\forall v, y_v(l') = 0$. Further, $\forall v, c, x_{v,c}(l') = 0$ from (10.6a).
2. For any quality level l', if $Y(l') = 1$, then from the definition of $y_v(l)$ and $Y(l)$, $\exists! v' \in V$, such that $y_{v'}(l') = Y(l') = 1$. Recall that $C(l') = \{c : X_c(l') = 1\}$ represents the set of clients that are assigned data with quality level l' by Step 1 in §10.2.
   a) For client $c' \in C(l')$, $X_{c'}(l') = 1$. Since v' is the unique data center set across V such that $y_{v'}(l') = 1$, from (10.6a) and (10.6b), $x_{v',c'}(l') = 1$ and $x_{v,c'}(l) = 0$, $\forall v \neq v'$ or $l \neq l'$. In other words, $x_{v,c'}(l') = y_v(l')$, $\forall v \in V, c \in C(l')$.
   b) For client $c \notin C(l')$, $X_c(l') = 0$. From the definition of $X_c(l')$, $x_{v,c}(l') = 0$, $\forall v$.

In all above cases, the optimal solution $\{x_{v,c}(l), y_v(l)\}$ of (10.6) satisfies the following:

$$x_{v,c}(l) = \begin{cases} y_v(l), & \text{if } c \in C(l), \\ 0, & \text{otherwise.} \end{cases} \tag{C.11}$$

Next, we use this form for $x_{v,c}(l)$ to derive $y_v(l)$. After substituting (C.11) into (10.6), most constraints become trivial due to the form of (C.11) and the optimality of $X_c(l)$ and $Y(l)$. And we only need to optimize the objective function with the constraints stating that $y_v(l)$ is binary, and $\sum_v y_v(l) = Y(l)$.

Thus, we only need to optimize the following problem.

$$\begin{aligned}
\text{minimize} \quad & \sum_{l:Y(l)=1} \sum_{v=1}^{V} \beta_v(l) y_v(l) \\
& + \sum_{l:Y(l)=1} \sum_{c \in C(l)} \sum_{v=1}^{V} (\alpha_{v,c}(l) + f(l)) y_v(l) \\
\text{subject to} \quad & \sum_{v=1}^{V} y_v(l) = Y(l), \forall v, c, l \\
& y_v(l) \in \{0,1\}, \forall v, c, l
\end{aligned}$$

The above optimization can be decoupled by l and optimized across v, yielding the following closed form solution.

$$y_v(l) = \begin{cases} 1, & \text{if } Y(l) = 1 \text{ and} \\ & v = \text{argmin}\{\beta_v(l) + \sum_{c \in C(l)} \alpha_{v,c}(l)\}, \\ 0, & \text{otherwise.} \end{cases} \tag{C.11}$$

C.3 Bulk Data Contracting

In bulk data contracting, the data cloud only has to pay a one-time fee $f(l,p)$ for data $q(l,p)$, no matter how many times the data is replicated on the cloud and transferred to clients. Compared to per-query contracting, the main difference lies in the purchasing fees modeling. Defining $z(l,p) \in \{0,1\}$ to be equal to 1 if and only if data of quality $q(l,p)$ from data provider p is transferred to the data cloud, the whole optimization problem can still be formulated in a form similar to (9.5), with the purchasing costs now given by (9.4) and with the addition of the following constraint:

$$y_{p,d}(l) \leq z(l,p), \ \forall c, l, p, d \tag{C.12}$$

This constraint states that any data placed in the data cloud must have been purchased by the data cloud. As in the per-query contracting case, the data purchasing/placement decision for data from one data provider does not impact the data purchasing/placement decision for any other data providers. Thus, we drop the index p in the following.

In general, the cost minimization problem for bulk contracting is NP-hard. To be specific, the 1-level UFLP can reduce to the cost minimization problem for

a geo-distributed data cloud, and the cost minimization problem can reduce to the 2-level UFLP in the bulk case. In the 2-level UFLP, facilities are organizing on 2 levels, $J_1 \times J_2$; each customer $i \in I$ has to be assigned to a valid path $p \in J_1 \times J_2$. A pass is valid if and only if both facilities are open along the path. More details on the 2-level UFLP can be found in [97].

The first reduction follows directly from the first part of the proof for Theorem 8. It can be easily proved by defining facilities in J_1 to be the quality levels, and using the same reformulation as the second part of the proof for Theorem 8 for the facilities in J_2, i.e. define facilities inJ_2 to be pairs of quality levels and data centers. In the reduction, a facility $j_1 \in J_1$ is open if and only if the corresponding quality level l is purchased, and a facility $j_2 \in J_2$ is opened if and only if data of quality level l is placed in data center d.

While the cost minimization in bulk contracting is generally hard, it can be solved optimally in both the single data center and the geo-distributed data cloud settings under certain assumptions.

For the single data center case, we always have $z(l) = y(l)$ for all quality level l - this follows immediately from dropping the dependence of $y_d(l)$ in d, implying that $z(l)$ is only lower-bounded by $y(l)$ in the constraints. Furthermore, if the execution costs are the same across quality levels, the cost minimization problem can be formulated as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{l=1}^{L} \left(\beta(l) + f(l)\right) y(l) \qquad \text{(C.13)} \\ \text{subject to} \quad & x_c(l) \leq y(l),\ \forall c, l \\ & \sum_{l=w_c}^{L} x_c(l) = 1,\ \forall c \\ & x_c(l) \geq 0, \forall c, l \\ & y(l) \geq 0,\ \forall l \\ & x_c(l), y(l) \in \{0, 1\}, \forall c, l \end{aligned}$$

Since the decisions for variables $\{x_c(l)\}$ do not affect the objective value, (C.13)

can be written as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{l=1}^{L} \left(\beta(l) + f(l)\right) y(l) \\ \text{subject to} \quad & \sum_{l=w_c}^{L} y(l) \geq 1, \ \forall l, c \\ & y(l) \in \{0, 1\}, \forall l \end{aligned} \tag{C.14}$$

Since there are customers buying the highest quality level, the highest level quality L is always purchased by the data cloud and $y(L) = 1$ in any feasible solution. Since all customers are satisfied and all costs are non-negative, an optimal solution for (C.14) is $y(L) = z(L) = 1$, $x_c(L) = 1$ with all other variables are set to 0. The result implies the data cloud will only purchase the highest quality level of data and serve that data to every customers.

For a geo-distributed data cloud, the cost minimization problem is generally hard. However, if we assume the operation cost and execution cost are independent of l, i.e., $\beta_d(l) = \beta_d$ and $\alpha_{d,c}(l) = \alpha_{d,c}$, it is easy to show that the optimal solution will only purchase the highest quality data as in the single data center case. We can then use Step 2 in §10.2 to give an optimal solution to the data placement problem.