Efficient Resource Placement in Cloud Computing and Network Applications

Yuval Rochman
Tel Aviv University
Tel-Aviv, Israel
yuvalroc@post.tau.ac.il

Hanoch Levy
Tel Aviv University
Tel-Aviv, Israel
hanoch@cs.tau.ac.il

Eli Brosh
Vidyo
New Jersey, USA
eli@vidyo.com

Abstract

We address the problem of resource placement in general networking applications, in particular cloud computing. We consider a large-scale service faced by regionally distributed demands for various resources. The service aims at placing the resources across regions to maximize profit, accounting for demand granting revenues minus resource placement costs. Cloud computing and online services, utilizing regional datacenters and facing the problem of where to place various servers, naturally fall under this paradigm.

The main challenge posed by this setting is the need to deal with arbitrary multi-dimensional (high-dimensionality) stochastic demands. To allow the modeling of a large variety of applications we consider a fairly general framework: 1) Operation cost is an arbitrary semi-separable convex function, 2) Demand is either static or dynamic, and 3) Regional storage capacities can be either bounded or not. We show that, despite the challenging stochastic combinatorial complexity, one can optimize the system operation using fairly efficient algorithms. Our framework deals with the issues of static placement, dynamic placement and capacity allocation and provide efficient algorithms to each of them. Our solutions are based on new theoretical techniques using graph theory methodologies that can be applied to other optimization/combinatorial problems. Our analysis reveals that the length of the demand distribution tail significantly affects the amount of resources needed in the system.

I. Introduction

Cloud computing has emerged as an attractive solution for delivering large-scale online services such as web services, streaming and social network applications. Existing cloud computing platforms like Amazon EC2 [1] and Microsoft Azure [2] typically organize a shared pool of servers in geographically distributed datacenters to provide on-demand delivery of computing resources at scale. Large-scale services often feature demands that are highly dynamic and geo-distributed and leverage cloud computing to serve users more efficiently and reliably.

A common approach used by large-scale services is to place (server) resources at various geographical areas to be close to their users and guarantee adequate levels of service quality, e.g., low response times. It is typically preferred to serve a demand by a resource located in the same area rather than by a remotely located resource. At the same time, placing and maintaing a resource incurs a cost such as server rental cost. For example, Amazon EC2 offers several server instance types natively and few thousands more through a marketplace[1]. It bills instance usage on a pay-per-use basis, e.g., according to an on-demand price plan that varies with instance type and datacenter location. Thus, engineering such services in a cost-effective manner is far less than trivial.

Large-scale service providers face the challenging problem of how to place resources over the areas so as to minimize the total operation cost while providing adequate level of performance, taking into account the demand dynamics. For sake of generality, we capture demand as a multi dimensional distribution consisting of the demand distribution for each resource and in each area. This problem typically involves solving a joint placement-assignment problem: 1) deciding on the number of resources (of each type) in each area, and then 2) assigning demands efficiently to the resources. Unlike many resource placement problems, our joint problem is driven by a stochastic demand.

Our goal is to provide a framework and algorithms to address this challenge. To this end, we develop a general model that captures the major parameters that affect the design of service placement algorithms in geo-distributed environments, including: 1) A variety of resource types, 2) A stochastic demand, 3) The cost of placing and maintaining a resource, 4) The benefit of satisfying the demand, e.g., the revenue associated with reducing user response times, and 5) The area capacity and locations. The problem is of high dimensionality due to the amount of areas, amount of resource types and, especially, the need to account for (multi-dimensional) full probability distribution functions of the demand.

A. Model Generality, Structure of the Paper and Contributions

The generality of the framework allows solving, in addition to service placement problems, a large variety of distributed resource allocation problems. One example is a cloud provider that offers infrastructure as a service (e.g., server instances), and maintains geo-distributed datacenters to be close to its users. It is typically better to provide a request for a server resource from a local datacenter as apposed to a remote one. The provider's objective is to place servers (optimally) across datacenters and in the longer time-scale find the right sizing of the datacenters, so as to optimize the total operating cost. Call center operators face similar challenges of how to allocate the available personnel across the contact centers (e.g., on a daily basis), and how to estimate the total personnel pool (e.g., on a yearly basis). Another related example is that of content distribution networks. Such systems feature large volumes of content and highly-diverse demands, and seek to place the right content replicas at the edge servers so as to satisfy the demand in a cost-effective manner.

Our paper makes the following contributions: First, we propose an algorithmic solution to the optimal placement problem (Section III) and demonstrate that, despite the problem's high dimensionality, it runs in **relatively low**

[1]An EC2 instance is a virtual machine that runs a specific application image.

complexity ($O(smk)$, where s is system total capacity, m number of resource types, and k number of regions). Our solution is based on presenting a closed-form revenue-based formulation and proposing a reduction to a min-cost flow problem (Sections IV, V); Solution of the min-cost flow problem is then offered (Section VI). Compared with [3] our solution is **tailored for cloud computing applications**, whereas the solution in [3] does not account for the major parameters relevant to cloud computing, such as server rental costs, server capacity and resource type dependency. Thus, the problems addressed in this paper are harder than the one presented in [3]. We propose new solutions techniques such as Successive-Shortest-Path-Negative-Edges combined with node-potentials; these techniques yield a low complexity algorithm which is significantly **faster** than the one proposed in [3].

In addition, this paper addresses two different problems which are not addressed in [3]: the capacity planning problem and the dynamic placement problem, as described next. In Section VII we introduce an algorithm that solves the *unbounded* placement problem where there is no restriction on the number resources allowed in an area. The unbounded problem is useful for capacity planning – it allows one to determine the total quantity of resources (and their locations) that optimizes the service operating cost, i.e., when the marginal profit approaches zero.

To derive the unbounded placement solution we introduce a new theoretical result, **SSP convexity theorem**, that can be applied in other optimization problems where a min-cost flow algorithm called **Successive Shortest Path** (SSP) is used (e.g [4], [3], [5]).

In Section VIII we introduce and analyze a novel dynamic *online* placement algorithm that accounts for time-varying demand. We prove that small changes in the demand can lead to significant changes in the optimal placement, though not necessarily in the optimal cost. Our algorithm, called **LONA (Lazy ONline Algorithm)**, finds an approximate placement (close to the optimal one) by deciding whether to maintain the current placement or reconfigure the resources based on a pre-defined cost deviation threshold, avoiding unnecessary reconfigurations.

Finally (Section IX), we use our unbounded algorithm to evaluate the performance of the system by solving the capacity planning problem. We demonstrate two major results: 1) As the demand for esoteric resources decreases, the size of the solution as well as the number of iterations required by the algorithm increases. 2) Using stopping rules differing from the one proposed in Section VII, such as stopping whenever the marginal cost decreases below some threshold, is inefficient.

The reader should note that the framework addressed in this paper is fairly wide and entails, in some cases, many details; for clarity of presentation, we placed some of these details in the appendix.

II. Related Work

The problem of resource placement in distributed systems often falls under facility location theory [6]. This area has received significant attention from the viewpoint of both analysis and algorithmic solutions. Our version of the problem differs from traditional facility location problems in that it incorporates stochastic demands and capacity constrains on the locations (servers).

Early works on distributed resource placement primary focused on placing content replicas across a content distribution or a web cache network (see e.g. [7], [8]). However, most of these works have focused on static or deterministic demand profiles, paying little attention to the capacity limits at individual servers and geographical areas.

With the growth of cloud computing and large-scale online services, the problem of server placement in geo-distributed environments has received increasing attention. Some works studied the problem from a standpoint of a service provider. For example, [9] focused on dynamically optimizing service placement while ensuring performance requirements; and [10] studied algorithms for dynamic scaling of social media applications. Other works looked at the problem from a cloud provider's viewpoint, i.e., assigning workloads to distributed datacenters in a cost-effective manner [11]. These works typically develop an optimization problem with deterministic inputs and apply it periodically. In contrast, we provide optimal algorithmic solutions that account for the full demand distribution. Further, our dynamic algorithm can approximate the optimal solution by a given threshold, avoiding the need for demand prediction mechanisms, as done in many other works.

A variant of the placement problem which accounts mainly for service costs (see basic model in Subsection III-A) has been considered in the context of content replication in P2P systems. [12] was perhaps the first to study a network setting similar to ours with an exponentially expanding topology for file sharing systems. It proved the optimality of proportional replication i.e., one based on the mean demand, under the assumption of abundant storage and upload capacity where all possible requests are always be served. In contrast, our model allows for restricting the number of resources (files), resulting in min-cost flow based replication.

Other relevant works are [13], [14], [15] that focused on P2P VoD replication systems. [13] proposed an optimal replication algorithm, called RLB, based on the assumption that the number of movies is much smaller than the number of peers. The work focused on small-scale networks, and do not consider geo-distributed topologies. [14] proposed placement framework for large-scale VoD service based on mixed integer program. While their model accounts for arbitrary demand pattern and network structure it assumes deterministic demand, whereas we consider stochastic one. Recently, [15] characterized the service efficiency of distributed content platforms as function of servers' storage size by using an asymptotic performance model for online matching algorithms. In this context, our work maps to a single content server storage model, which enables us to solve the combined optimization problem of matching and placement using exact analysis.

This work solves a wide and general setting; this is in contrast to [3], which formulated and treated the placement problem under a basic and limited model (reviewed in Subsection III-A). More differences between [3] and our paper are discussed in Subsection I-A.

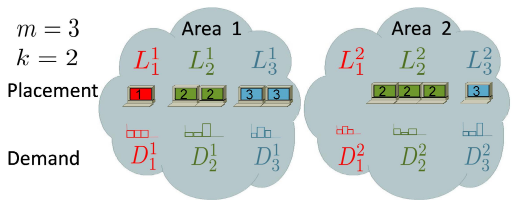

Fig. 1. An example of the system topology. Note that the storage is $s^1 = 5$ and $s^2 = 4$ and the placement is $L_1^1 = 1$, $L_1^2 = 0$, $L_2^1 = 2$ etc.

III. The model and the problem

For the sake of exposition, we start by describing a basic model for the placement problem with a simplified cost model. We then describe the full model and the optimization problem. Although we focus our examples on service placement in clouds, the generality of our model allows solving a variety of resource allocation problems spanning from server placement in server farms to personnel placement in call centers.

A. A basic model

We consider a system consisting of k **areas** indexed by $1, 2, \dots, k$. In each area one can place multiple resources of different **types** indexed by $1, 2, \dots, m$. Let L_i^j denote the number of type i resources placed in area j. The set of resources (servers) placed in these areas is called a **placement** and denoted by $L = \{L_i^j\}$. We assume that area j is associated with a storage value s_j representing a bound on the total number of resources that can be placed in it. Placement L is called **feasible** if the number of resources in area j is not larger than s^j, i.e $L^j := \sum_{i=1}^{m} L_i^j \le s^j$.

We consider a stochastic demand for resources. Let D_i^j be a random variable denoting the number of requests for type-i resource in area j. We do not make any assumption on the distribution of D_i^j, namely it can be of an **arbitrary** (non-negative) **distribution**. We assume that the demand CDF values $\Pr(D_i^j \ge n)$, $n = 0, 1, 2, \dots$, are calculated in $O(1)$, and are available in an external data base. An example of the topology of a system is depicted in Fig. 1, where every computer represents a resource.

Service cost model. Consider a request made in area j and a placement L. If the request is assigned to a resource in L, then the request is called **satisfied**. If the request is satisfied, it is assigned to either of: 1) A resource of L located in area j (and therefore the request is granted **locally**). 2) A resource of L located in a different area (granted **remotely**). If a request is not assigned to any resource in L, then it is called an **unsatisfied** request. The costs of a locally satisfied request, a remotely satisfied request, and an unsatisfied request are denoted by $C^{loc}, C^{rem}, C^{unsat}$, naturally obeying $C^{loc} \le C^{rem} \le C^{unsat}$. For example, in cloud-based applications, these costs can represent the revenue loss associated with increasing user response times when a request is granted remotely rather than locally or when its not granted at all.

Given a placement L and a deterministic realization d_i^j of the demand D_i^j, one can derive an optimal assignment of the resources to the demand, yielding minimal assignment cost (see Section IV). Let $C(L, d_i^j)$ be the optimal assignment cost and denote by g_{loc}, g_{rem} and g_{unsat} the corresponding number of requests granted under the optimal assignment from a local area, granted from a remote area, and unsatisfied ones, respectively. Then, the minimal assignment cost is simply $C(L, d_i^j) = C_{loc} \cdot g_{loc} + C_{rem} \cdot g_{rem} + C_{unsat} \cdot g_{unsat}$. The expected **service cost** $E|_D(C(L))$ is defined as the expected value of the minimal assignment cost over all demand realizations: $E|_D(C(L)) = \sum_{i=1}^{m} \sum_{j=1}^{k} C(L, d_i^j) \cdot \Pr(D_i^j = d_i^j)$.

B. *The full model*

The full model generalizes the basic model by capturing common properties of current cloud and network applications, such as resource placement costs and resource capacities.

Placement cost model. We associate a **resources cost** with placement L, denoted by $C_r(L)$, which represents the cost of placing and operating the resources of L. We assume that $C_r(L)$ is a semi-separable function. Roughly speaking, this means that $C_r(L)$ consists of the sum of individual functions, each of them is a function either of: 1) The number of resources placed of type i (L_i), 2) the number of resources placed in area j (L^j), 3) The number of type i resources placed in area j (L_i^j). A formal definition of semi-separable functions is given in Section IV-C.

This cost model allows us to capture a variety of server rental plans. For example, the on-demand server pricing of cloud providers such as EC2 can be captured as p_i^j, a fixed price for running type-i server in area-j, yielding a resource cost of $C_r(L) = \sum_{i=1}^{m} \sum_{j=1}^{k} L_i^j \cdot p_i^j$. A more complicated example can incorporate software licensing costs and area-specific charges. Let r_i be the licensing cost associated with type-i server and h^j be an add-on provisioning cost for area j. Then, the (semi-separable) resource cost becomes

$$C_r(L) = \sum_{i=1}^{m} \sum_{j=1}^{k} L_i^j \cdot p_i^j + \sum_{j=1}^{k} L^j \cdot h^j + \sum_{i=1}^{m} L_i \cdot r_i. \tag{1}$$

We assume that the resource cost is independent of the demand distribution D_i^j and of the service cost constants. We define the **total placement cost** $C_p(L)$ as the sum of the resource cost and the service cost

$$C_p(L) = C_r(L) + E|_D(C(L)). \tag{2}$$

We assume that $C_p(L)$ is a convex function. That is, that the marginal profit from adding a resource (see Remark 3.1) decreases as the quantity of resources increases, as is often the case in operating costs of real-world systems.

Resource capacity. A type-i resource can serve at most B_i requests. This parameter can reflect the upload or processing capacity of a server. Note that in the basic model $B_i = 1$.

Resource type dependency. In some applications, the service cost is dependent on the type of a request. For example, some server instances of a cloud provider may be better at handling requests than others. Then, the service cost of a request granted locally, remotely, and an unsatisfied request becomes $C_i^{loc}, C_i^{rem}, C_i^{unsat}$ respectively, where i is the request type. These costs obey $C_i^{loc} \leq C_i^{rem} \leq C_i^{unsat}$.

C. The placement problem

We define the (bounded) placement problem as a minimization of the (total) placement cost under storage constrains. More formally, given the demand distributions $\{D_i^j\}$, $i = 1, \ldots, m$, $j = 1, \ldots, k$, service cost parameters $C_i^{loc}, C_i^{rem}, C_i^{unsat}$, placement cost parameters, area storage values $s^1, s^2, \ldots s^k$, resource capacities B_i and an optimal matching algorithm, determine the optimal resource placement $L = \{L_i^j\}$, $i = 1, \ldots, m$, $j = 1, \ldots, k$, that minimizes the placement cost $C_r(L) + E|_D(C(L))$ among all feasible placements obeying $L^j = \sum_{i=1}^m L_i^j \leq s^j$.

Remark 3.1: In some applications service cost parameters are alternatively replaced by **service revenues parameters** $R^{loc} \geq R^{rem} \geq R^{unsat}$, representing the revenue of satisfying the demand. Under this setting we may define an equivalent problem of **maximizing** placement profit where placement cost is replaced by *placement profit* $E|_D(C(L)) - C_r(L)$.

The unbounded placement problem In this formulation we solve the problem of finding the minimal cost placement, where the number of resources in every area is **unbounded** (i.e., $s^j = \infty$ for all areas j). Note that the unbounded problem has an infinite number of feasible placements, making the modeling and solution harder. Under this setting, we assume that an optimal placement does exist. That is, there exists a finite placement L_{opt} such that for every placement L' $C_p(L_{opt}) \leq C_p(L')$.

The unbounded problem is useful for capacity planning, namely, to find how many resources are needed (and their locations) to optimize the total service operation cost.

D. Repositioning costs and the dynamic placement problem

In the dynamic placement problem, we assume that the demand is time-dependent where the demand distribution at time slot t is denoted as $D(t) = \{D(t)_i^j\}$. The total placement cost of placement L, which depends on the demand, $D(t)$, is denoted by $C_p^{D(t)}(L)$ and the optimal placement with respect to $D(t)$ is denoted by $L_{opt}(D(t))$. We call algorithm A an **online algorithm** if it computes the placement at time t, $L_A(t)$, only as a function of past information; that is, it cannot use $D(t+1), D(t+2), \ldots$ Note that $L_A(t)$ is not necessarily the optimal placement $L_{opt}(D(t))$.

Two important costs are relevant to an online algorithm: 1) The **total placement cost deviation**, and 2) The **resource repositioning cost**. The former, denoted $C_{dev}(A, t)$, evaluates the deviation of the total placement cost between the output placement $L_A(t)$ and the optimal placement $L_{opt}(D(t))$ with respect to demand $D(t)$. That is $C_{dev}(A, D(t)) \doteq |C_p^{D(t)}(L_A(t)) - C_p^{D(t)}(L_{opt}(D(t)))|$. We denote the placement cost deviation of A as the worst-case placement cost deviation over all inputs, i.e $C_{dev}(A) \doteq \max_{D(t)} C_{dev}(A, D(t))$.

The repositioning cost evaluates the cost of repositioning the resources, namely of transforming placement $L_A(t)$ to the placement $L_A(t+1)$. The repositioning cost, denoted as $r(A)$, counts the number of added and removed resources, and therefore the repositioning cost is the L_1 **distance** between $L_A(t)$ and $L_A(t+1)$, i.e $r(A,t) \doteq \sum_{i=1}^{m}\sum_{j=1}^{k} |L_A(t)_i^j - L_A(t+1)_i^j|$ ($L_A(t)_i^j$- number of type-i resources in region j). Finally, we denote the repositioning cost of A as the worst-case repositioning cost over all inputs, i.e. $r(A) \doteq \max_t r(A,t)$.

The goal of the dynamic placement problem is to develop an online algorithm whose repositioning cost and total cost deviation are low.

IV. The loss function

In this section we introduce the *loss* function, whose properties are a key for solving the placement problem. We prove the following fundamental claims: 1) A placement L solves the placement problem if and only if it minimizes the *loss* function over all feasible placements. 2) The loss can be computed using a simple closed-form formula. 3) The loss is a convex and a semi-separable function.

A. The loss function: preliminaries

The placement cost function as presented in Equation (2) is difficult to optimize. A key element in our analysis is a transformation of the (expected) service cost from that equation, $E_{|D}(C(L))$, into "differential revenue", $E_{|D}(R(L))$, which expresses the service cost in relative terms (e.g. cost of being served locally relatively to being served remotely). The transformation facilitates the solution while preserving the problem; its details are given next. To define the loss function, we use the following definitions:

Definition 1: Let M be a matching between a placement $L = \{L_i^j\}$ and a demand realization d_i^j. Let $g_i^{loc}(M)$, $g_i^{rem}(M)$ respectively denote the number of type-i requests granted locally and remotely under the matching algorithm M. The number of type-i requests granted **globally** (i.e, remotely or locally) is denoted by $g_i^{glo}(M) \doteq g_i^{rem}(M) + g_i^{loc}(M)$. We define the **local differential revenue constant** as the revenue gained from granting a type-i request locally compared to granting it remotely, i.e $R_i^{loc} \doteq C_i^{rem} - C_i^{loc} \geq 0$. The **global differential revenue constant** is the revenue gain from granting a type-i request remotely compared to not satisfying the request, i.e $R_i^{glo} \doteq C_i^{unsat} - C_i^{rem} \geq 0$. The *matching revenue* of M, denoted by $R(L, d_i^j, M)$, is defined as $R(L, d_i^j, M) = \sum_{i=1}^{m}(R_i^{glo} \cdot g_i^{glo}(M) + R_i^{loc} \cdot g_i^{loc}(M))$. The **maximal revenue value** $R(L, d_i^j)$ is the maximum value of the matching revenues $R(L, d_i^j, M)$ over all possible matchings M. We define the **service revenue** as the expected maximal revenue value, over all demand realizations, i.e $E|_D(R(L)) = \sum R(L, d_i^j) \cdot \Pr(D_i^j = d_i^j)$.

To this end, we define **the loss function** or simply **the loss** of placement L as the resource cost minus the service revenue,

$$loss_p(L) = C_r(L) - E|_D(R(L)). \tag{3}$$

The following claim proved in the appendix establishes the loss as a key for solving the placement problem:

Claim 4.1: For every placement L, the difference between the loss function and the total placement cost, $loss_p(L) - C_p(L)$ ($C_p(L)$ defined in Eq (2)), equals to $\sum_{i=1}^{m} E(D_i) \cdot C_i^{unsat}$, which is constant. Thus, a placement L solves the placement problem iff the placement minimizes $loss_p(L)$ among all feasible placements obeying $L^j = \sum_{i=1}^{m} L_i^j \leq s^j$.

B. A closed-form formula for the loss function

To optimize the loss function (Eq. (3)) one must: 1) Account for all possible demands, 2) Derive for each of them the optimal assignment of resources to demands, 3) Compute the cost of each such assignment, and finally 4) Derive the expected cost as computed over all these assignments. All these steps are conducted in our Assignment algorithm which is based on greedy matching principles and whose details (and proofs) are given in the appendix.

The Assignment Algorithm maximizes the number of locally granted requests in every area, and then it grants remotely the other unsatisfied requests. We prove in the appendix that given a placement $L = \{L_i^j\}$ and a demand realization d_i^j the algorithm grants globally $g_i^{glo} = \min(B_i \cdot L_i, d_i)$ (B_i is the type-i resource capacity) and grants locally $g_i^{loc} = \sum_{j=1}^{k} \min(B_i \cdot L_i^j, d_i^j)$ requests of type-i. We prove that the algorithm has a maximal revenue value which equals to

$$R(L, d_i^j) = \sum_{i=1}^{m} [R_i^{glo} \cdot \min(B_i \cdot L_i, d_i) + R_i^{loc} \cdot \sum_{j=1}^{k} \min(B_i \cdot L_i^j, d_i^j)]. \quad (4)$$

This yields a closed-form formula for the service revenue $E|_D(R(L))$, which is the expected value of the maximal revenue value. Thus Eq (3) and Eq (4) imply the following corollary:

Corollary 4.2: The loss of placement L equals

$$loss_p(L) = C_r(L) - \sum_{i=1}^{m} R_i^{glo} \cdot E|_{D_i}(\min(B_i \cdot L_i, D_i)) - \sum_{i=1}^{m} R_i^{loc} \cdot [\sum_{j=1}^{k} E|_{D_i^j}(\min(B_i \cdot L_i^j, D_i^j))]. \quad (5)$$

C. Properties of the loss formula

In order to find a placement that minimizes the loss, we study properties of the loss function, and use the following definitions:

Definition 2: Let $f : N^{m \cdot k} \to R$ be a discrete function.

1) f is called **semi-separable** iff there exists a set of discrete functions $\{g_i^j\}, \{g_i\}, \{g^j\}$ such that f is expressed by the following formula

$$f(L_1^1, L_2^1, \ldots) = \sum_{i=1}^{m} \sum_{j=1}^{k} g_i^j(L_i^j) + \sum_{i=1}^{m} g_i(L_i) + \sum_{j=1}^{k} g^j(L^j), \quad (6)$$

where $L_i = \sum_{j=1}^{k} L_i^j$, and $L^j = \sum_{i=1}^{m} L_i^j$. The set $M_g(f) = \{g_i^j\} \bigcup \{g_i\} \bigcup \{g^j\}$ is called the **marginal functions** of f.

2) Given a one-dimensional discrete function $g : N \to R$, define its **differential function** Δg as the difference between its successive values, i.e $\Delta g(n) = g(n) - g(n-1)$ for $n \geq 1$. The function g is called **convex** if its differential function Δg is monotonically non-decreasing, i.e, $\Delta g(n) \geq \Delta g(n+1)$ for all $n \geq 1$.
3) The n-dimensional discrete function $f : N^n \to R$ is called **convex** if for every vector $v^{-i} \in R^{n-1}$ the partial function $g_i^{v^{-i}}(x) = f(v_1, v_2, \ldots, v_{i-1}, x, v_{i+1}, \ldots)$ is convex.
4) $f : N^n \to R$ is called **concave** if $-f$ is convex.

In Section III we assumed that the resource cost, $C_r()$, is a semi-separable function. This means that there exists a set of marginal functions $M_\zeta(C_r) = \{\zeta_i^j\} \bigcup \{\zeta_i\} \bigcup \{\zeta^j\}$ such that

$$C_r(L) = \sum_{i=1}^{m} \sum_{j=1}^{k} \zeta_i^j(L_i^j) + \sum_{i=1}^{m} \zeta_i(L_i) + \sum_{j=1}^{k} \zeta^j(L^j). \tag{7}$$

In the linear resource cost example presented in Eq (1) the marginal functions are simply $\zeta_i^j(L_i^j) = L_i^j \cdot p_i^j, \zeta^j(L^j) = L^j \cdot h^j$ and $\zeta_i(L_i) = L_i \cdot r_i$.

In the appendix we prove the following claim:

Claim 4.3: The loss function $loss_p$ is convex and semi-separable.

Thus, there exists a set of marginal convex functions $M_\gamma(loss_p)$ such that $loss_p(L) = \sum_{i=1}^{m} \sum_{j=1}^{k} \gamma_i^j(L_i^j) + \sum_{i=1}^{m} \gamma_i(L_i) + \sum_{j=1}^{k} \gamma^j(L^j)$. To prove that the loss is convex, we use the assumption in Section III that the placement cost $C_p()$ is convex, and Claim 4.1.

V. Reduction to a min-cost flow problem

As shown in the last section, the optimal solution for the placement problem must minimize the loss function (Eq (3)), a semi-separable and convex function. In this section we show how to reduce the placement problem to a **min-cost flow** problem using the **marginal-differential functions**.

A. Introduction to the min-cost flow problem

We start describing the **min-cost flow problem** [16], which is a generalization of the notable max flow problem. In this problem one considers a directed graph $G = (V, E)$ where every edge $e \in E$ has integer **capacity** $c(e)$ and a real-value **weight** $w(e)$ (alternatively, called **cost**). The graph must contain two different nodes: a source node x and a sink node y. An **x-y-flow** $f : E \to R^+$ is defined on the graph edges $(v, v') \in E$ in the same way as defined in the max-flow problem and must satisfy: 1) **Capacity constraint:** for each edge $e \in E$, $0 \leq f(e) \leq c(e)$. 2) **Conservation of flows:** for every vertex $v \in V \setminus \{x, y\}$ we have $\sum_{(v',v) \in E} f(v', v) = \sum_{(v,v') \in E} f(v, v')$. In addition to the standard definitions, we define the **flow at node** $v \neq x, y$ as the income flow (and by conservation of flows, the outcome

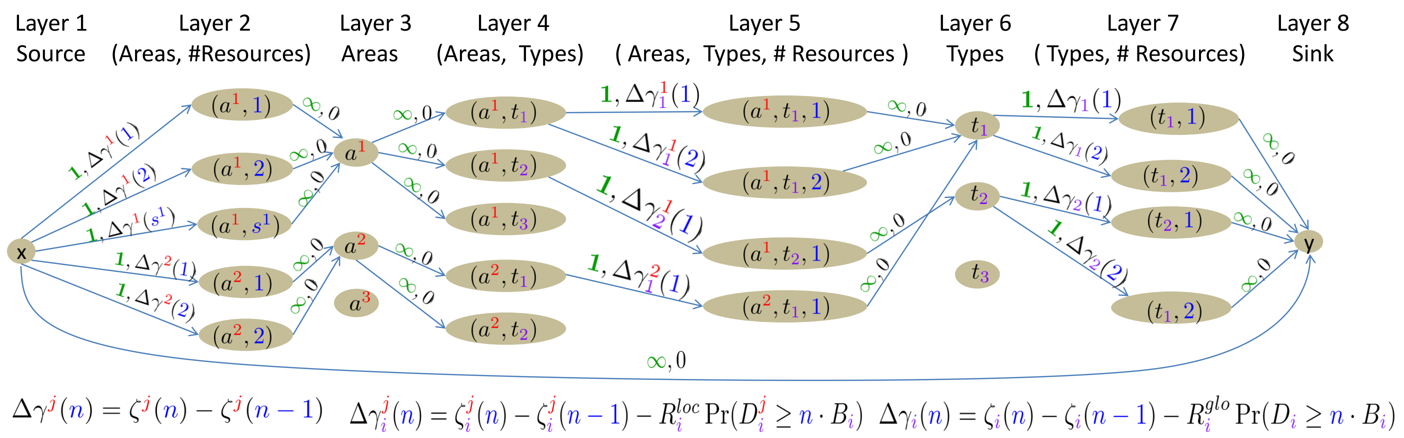

Fig. 2. The 8-layer graph . Note that the edge weights, which are the marginal-differential functions defined in Claim 5.1, can be negative.

flow) to (from) node v. We denote it by $f^{in}(v)$, which equals $f^{in}(v)=\sum_{(v,v')\in E}f(v,v')(=\sum_{(v',v)\in E}f(v',v)=f^{out}(v))$. The **flow value** of f, as defined in the max-flow problem, is $|f|=\sum_{(x,v)\in E}f(x,v)=\sum_{(v,y)\in E}f(v,y)$. The **weight (or cost)** of flow f is $w(f)=\sum_{e\in E}f(e)w(e)$.

The classic **min-cost flow** problem with required flow $|f|=k$ is to find a flow $f_{opt}(k)$ of value k that has minimal weight among all flows of value k. This means that for every flow f' such that $|f'|=|f_{opt}(k)|=k$ we have $w(f_{opt}(k))\leq w(f')$.

B. Reduction of the placement problem to a min-cost flow problem

Given the loss function $loss_p$ with its marginal functions $M_\gamma(loss)$, we define the **marginal-differential** functions $\{\Delta\gamma_i^j\}\bigcup\{\Delta\gamma_i\}\bigcup\{\Delta\gamma^j\}$ simply as the differential of the correspond marginal functions $\{\gamma_i^j\}\bigcup\{\gamma_i\}\bigcup\{\gamma^j\}$. The following claim regarding the marginal differential functions is straightforward:

Claim 5.1: The marginal-differential functions $\Delta\gamma$ can be calculated using the following equations:

$$\Delta\gamma^j(n)=\zeta^j(n)-\zeta^j(n-1) \tag{8a}$$

$$\Delta\gamma_i^j(n)=\zeta_i^j(n)-\zeta_i^j(n-1)-R_i^{loc}\Pr(D_i^j\geq n\cdot B_i) \tag{8b}$$

$$\Delta\gamma_i(n)=\zeta_i(n)-\zeta_i(n-1)-R_i^{glo}\Pr(D_i\geq n\cdot B_i). \tag{8c}$$

In a linear resource cost function (Eq (1)) the marginal-differential functions are equal to $\Delta\gamma^j(n)=h^j$, $\Delta\gamma_i^j(n)=p_i^j-R_i^{loc}\Pr(D_i^j\geq n\cdot B_i)$ and $\Delta\gamma_i(n)=r_i-R_i^{glo}\Pr(D_i\geq n\cdot B_i)$.

Remark 5.2 (Properties of the marginal-differential functions): The marginal-differential $\Delta\gamma(n)$ are 1) monotonically increasing, which stems from the fact that the loss is convex (see Claim 4.3), and 2) are negative if the addition the nth resource benefits the system i.e, has a negative marginal cost.

We reduce the placement problem to a min-cost problem using the graph given in Fig. 2 as follows: We define a directed 8-layer graph $G^8 = (V^8, E^8)$. On every edge a pair $c(e), w(e)$ so that $c(e)$ is the capacity function (presented in olivegreen color) and $w(e)$ is the weight function. The graph is composed from the following 8 layers: The **source** x, the **(area, #resources)** layer, the **area** layer, the **(area, type)** layer, the **(area, type, #resources)** layer, the **type** layer, the **(type, #resouces)** layer, and finally the **sink** y. We also connect the source x to the sink y.

In the graph we denote area j by a^j, type i by t_i, and #resouces by a number. The (area, #resouces), (area, type, #resources) and (type, #resources) layers, for example, are respectively composed of nodes (a^j, n) (where $1 \leq j \leq k$ and $1 \leq n \leq s^j$), (a^j, t_i, n) (where $1 \leq j \leq k$, $1 \leq i \leq m$,$1 \leq n \leq s^j$) and (t_i, n) (where $1 \leq i \leq m$ and $1 \leq n \leq \sum_{j=1}^{k} s^j$). The entering edges to nodes (a^j, x), (a^j, t_i, x) and (t_i, x) have respectively the marginal-differential weight $\Delta\gamma^j(x)$, $\Delta\gamma_i^j(x)$ and $\Delta\gamma_i(x)$ with capacity of 1. All other edges have weight 0 with capacity ∞.

Finding the min-cost flow on G yields the optimal placement as presented in the following lemma:

Lemma 5.3: Let f_{opt} be the min-cost flow in G^8 with a required flow of $|f| = \sum_{j=1}^{k} s^j \doteq s$. Let L be a placement with components equal to the flow in nodes (a^j, t_i), i.e $L_i^j = f_{opt}^{in}(a^i, t^j)$. Then L solves the placement problem.

The correctness of the lemma is presented in the appendix, and stems from the fact that the marginal-differential weights are monotonically increasing.

VI. Solving the min-cost flow problem

In order to solve the min-cost flow problem, one can use the classic Successive Shortest Path (SSP) algorithm with node potentials. The algorithm we introduce, called SSP-NE (Successive Shortest Path with Negative Edges), differs from the classic SSP, which cannot be used on the 8-layer graph G^8, where the edges weight $w()$ can be negative (Remark 5.2).

The SSP algorithm (as well as SSP-NE) operates in a greedy fashion: it find the best possibility of adding in every iteration a resource to a region, and then substitutes resources between different regions. It maintains in its i^{th} iteration an optimal placement for i resources.

A. *Classic SSP -preliminaries*

We begin by describing the classic SSP algorithm, and then present our SSP version. SSP is a well-known algorithm that solves the min-cost flow problem for general graphs with non-negative weights. Given a flow f on a graph $G = (V, E)$, the Successive Shortest Path (SSP) uses the **residual graph** $G_f = (V, E_f)$. On the residual graph edges one defines weight w_f and capacity c_f. Then, the residual graph G_f is constructed from graph G and from flow f by the following steps: 1) Add to G_f edges from G, such that edge $(v, v') \in E$ will have weight $w_f(v, v') = w(v, v')$ and capacity of $c_f(v, v') = c(v, v') - f(v, v')$. 2) Add the reverse edges of G. That means, if

$(v, v') \in E$, then add edge (v', v) to G_f with weight $w_f(v', v) = -w(v, v')$ and capacity of $c_f(v', v) = f(v, v')$. Note that for every edge e in G_f we have $c(e) \geq 0$. 3) For every edge $e \in E_f$ with capacity $c(e) = 0$ set its weight to be $w_f(e) = \infty$.

1) The shortest path algorithm building block and the use of node potentials: Shortest path algorithms are key building blocks for SSP. Two notable algorithms to calculate the shortest paths from one source v to all other vertices, are the **Bellman-Ford** and **Dijkstra's** algorithms. Dijkstra can run only on graphs with non-negative edges. Bellman-Ford can run also on graphs with negative edges (assuming it does not contain a negative cycle), but has a higher complexity than Dijkstra. The running time of Bellman-Ford on graph $G = (V, E)$ is $O(|E| \cdot |V|)$ compared to $O(|E| + |V| \cdot \log |V|)$ in Dijksra. More information on those algorithms can be found in [17].

One can use the **reduced weight optimality conditions** taken from [18] in order to prove that a given flow f is a min-cost flow. To show these conditions, we use the definition of **node potentials** $\pi : V \rightarrow R$ which is a function defined on the vertices of the residual graph $G_f = (V, E_f)$. Given the potential function π the **reduced weight** of an edge $(v_1, v_2) = e \in E_f$ is defined as $w_f^\pi(v_1, v_2) = w_f(v_1, v_2) - \pi(v_1) + \pi(v_2)$. A node potential function π satisfies the **non-negative reduced weight conditions** if the reduced weights of every edge $e \in E_f$ are non-negative (i.e $w_f^\pi(e) \geq 0$).

The reduced weight optimality conditions are presented in the following theorem. We will use it for proving the correctness of SSP-NE and for proving Theorem 8.2 in Section VIII.

Theorem 6.1 (Reduced Weight Optimality conditions): Let f be a flow with a required flow of $|f| = k$. Then f is a min-cost flow iff there exists a node potential function $\pi : V \rightarrow R$ that satisfies the non-negative reduced weight conditions.

2) A description of SSP: Finally, we present the classic SSP. In the initial step, SSP assigns the zero flow $f := 0$ (i.e $f(e) = 0$ for all $e \in E$) with flow value $|f| = 0$ and constructs the flow residual graph G_f. The algorithm sets the node potentials π to 0.

SSP works iteratively, and in the i^{th} iteration it calculates a min-cost flow of flow value larger than or equal to i (in the G^8 graph, it is exactly equal to i). SSP executes the following steps in every iteration: 1) Check if the flow value $|f|$ equals to the required flow k. If so, then f is the optimal flow and SSP terminates. 2) Use Dijksra's algorithm to obtain the shortest paths distances from the source x on G_f with respect to the non-negative reduced weights w^π. We retrieve the shortest path p between x and y, which is called the **augmenting path**. If the shortest path's weight is infinity (i.e $w_f^\pi(p) = \infty$) then the maximal flow value of the graph G is strictly less than the required flow k, and SSP returns an error. 3) We augment $\delta = \min(k - |f|, \min\{c(e) | e \in E\}) > 0$ units of flow through p. That means if $(v, v') = e \in p$ is in the original graph (i.e $e \in E$) then we update $f(e) \leftarrow f(e) + \delta$, and if the reverse edge in G (i.e $(v', v) \in E$) then we update $f(v', v) \leftarrow f(v', v) - \delta$. 4) The node potentials π are updated to be the previous node potentials minus the shortest path distance vector, i.e $\pi \doteq \pi - d$, where $d(v)$ is the shortest path between x and v. 5) A new residual graph G_f is created according to the updated flow f.

3) SSP in the context of resource allocation problem: One can notice that the the weight of the shortest path found by SSP over the 8-layer G^8 graph equals the cost of the best possibility for adding a resource to a region, and then substituting resources between different regions.

B. SSP-NE (Successive Shortest Path with Negative Edges)

The classic SSP cannot be used on weighted graph G with negative edges, and particularity cannot be used on the 8-layer graph G^8 (Remark 5.2). Thus we introduce the SSP-NE algorithm for solving the problem over a general subsets of instances, which includes the 8-layer graph G^8 .

As opposed to the classic SSP, in our version the node potentials of the vertices are not automatically set to 0 in the initial step. Rather, we use Bellman-Ford on G_f to calculate the shortest paths from source x with respect to the original weight function w_f. If $d(v)$ is the shortest path weight between x and a node v, then we set the initial node potential π to be $\pi(v) = -d(v)$. By using this settings, π satisfies non-negative reduced weight conditions, as shown by the following theorem (proved in the appendix).

Theorem 6.2: Let G be a graph with arbitrary edge weights w and with no negative cycles. In every iteration of SSP-NE, the node potential π satisfies the non-negative reduced weight conditions.

The 8-layer graph G^8 is acyclic, and in particular, it does not contain any negative cycle. Thus we can use SSP-NE on G^8 to retrieve an optimal placement L_{opt} using Lemma 5.3. The complexity of applying SSP-NE to the placement problem is $O(s^2m^2)$, where $s = \sum_{j=1}^{k} s^j$ be the total storage value m is the number of resource types. In the next section we present the **Bipartite Graph Algorithm** (BGA) , which solves the bounded capacity problem faster than SSP-NE ($O(skm)$, where k is the number of areas). Further, as opposed to SSP-NE, BGA can be used for solving the unbounded capacity problem.

VII. Capacity Planning – The Unbounded Placement Problem

In this section we are interested in dealing with capacity planning, which involves concurrently optimizing the number of resources used in the system and their placement. To this end we solve the unbounded placement problem, where the area storage is **unbounded** (i.e $s^j = \infty$ for all j).

The solution presented earlier for the bounded problem (using SSP on the 8-layer graph G^8) **cannot** solve the unbounded problem for two reasons: 1) The number of nodes in G^8 is $O(sm)$, where s is the total storage capacity of the system. Thus the unbounded problem, where $s = \sum_{j=1}^{k} s^j = \infty$, would require an infinite graph and it cannot find the shortest path in finite time. 2) The required flow for SSP equals to the total storage capacity, i.e $|f| = s = \infty$. Thus SSP, which runs $|f|$ iterations, will never stop.

To address the first problem, we propose the **Bipartite Graph Algorithm (BGA)**[2], which removes unnecessary

[2]The BGA version we provide in this article is different than the one suggested in [3]. For more information see Subsection I-A

nodes in the graph, transforming the graph to a finite one. To address the second problem we present a stopping rule called the **non-negative weight criterion**.

BGA imitates the behavior of SSP. That means, in every iteration the algorithm finds a shortest path, which is equivalent to adding a resource in a region and then substituting resources between different regions.

To show the non-negative weight criterion we present and prove **SSP convexity theorem**. The theorem, as far as we know, is a new contribution to graph theory and it can be used in optimization problems where the SSP algorithm is used. The SSP convexity theorem implies that once it is positive it will always be positive and no further improvement can be achieved by adding more resources.

A. The Bipartite Graph Algorithm (BGA)

To present BGA, we study the structure of the **monotone** shortest paths between two nodes in the 8-level residual graph G_f^8 (presented in Subsection VI-A).

Let f be a flow, and let v_i and v_j represent respectively a node in layer-i and a node in layer-j of G_f^8, for $1 \leq i \neq j \leq 8$. An edge $e = (v_i, v_j) \in G_f^8$ is called a **forward edge** iff $i < j$. If $e = (v_i, v_j)$ is not a forward edge, then it is called a **backward edge**. A path p in G_f^8 is called a **forward path** iff all the edges composing the path are forward edges. Similarly, a path p in G_f^8 is called a **backward path** iff all the edges composing the path are backward edges. A path p is **monotone** if it is a forward or a backward path.

The **bipartite-like graph**, denoted as $G_f^B = (V^B, E_f^B)$, is a sub-graph of the 8-layer residual graph G_f^8 constructed from the 8-layer graph G^8 (i.e $V^B \subseteq V^8$). The graph represents the shortest monotone paths in G_f^8, and is composed of the following layers: The first layer consists of the source node x, the second layer consists of area nodes a^j, the third layer consists of resource type nodes t^i, and the last layer consists of the sink node y. These are layers $1, 3, 6, 8$ in G_f^8. The graph, depicted in Fig. 3, resembles a bipartite graph, excluding the source and sink nodes.

The edge weight between u and v in the bipartite-like graph, which is denoted by $w_f(u \overset{\min}{\to} v)$, equals to the shortest monotone path between u and v in G_f^8. For example, if we denote $p(n)$ as the forward path composed from nodes a^j, (a^j, t_i), (a^j, t_i, n) and t_i in the 8-layer residual graph G_f^8, then the edge weight between a^j and t_i in G_f^B equals to the weight of the shortest monotone path, i.e. $\min_n w_f(p(n))$.

In the appendix we prove the following theorem:

Theorem 7.1: The shortest path in the bipartite-like graph G_f^B has the same weight as the shortest path in the 8-layer residual graph G_f^8.

As in SSP, BGA uses the node potentials technique (see Subsection VI-A1) and defines the reduced weight of an edge $u \overset{\min}{\to} v$ as $w_f^\pi(u \overset{\min}{\to} v) = w_f(u \overset{\min}{\to} v) - \pi(u) + \pi(v)$. The node potentials in BGA are identical to the potentials of the corresponding nodes in SSP. That means, if one denotes respectively π_i^{BGA} and π_i^{SSP} as the node

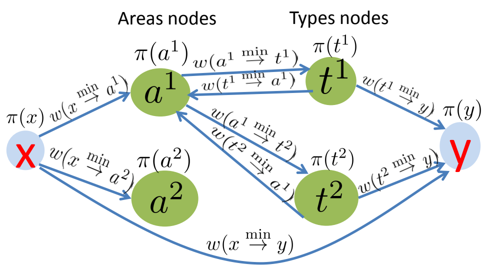
Areas nodes
Types nodes
$\pi(a^1)$
$w(a^1 \overset{\min}{\to} t^1)$
$\pi(t^1)$
$w(t^1 \overset{\min}{\to} a^1)$
a^1
t^1
$\pi(x)$
$w(x \overset{\min}{\to} a^1)$
$w(t^1 \overset{\min}{\to} y)$
$\pi(y)$
$w(a^1 \overset{\min}{\to} t^2)$
$w(t^2 \overset{\min}{\to} a^1)$
x
y
$w(x \overset{\min}{\to} a^2)$
$\pi(a^2)$
$\pi(t^2)$
$w(t^2 \overset{\min}{\to} y)$
a^2
t^2
$w(x \overset{\min}{\to} y)$

Fig. 3. The bipartite-like graph G_f^B; Node potentials π marked on the nodes.

potentials in the i-iteration of SSP and BGA then $\pi_i^{BGA}(v) = \pi_i^{SSP}(v)$ for all $v \in V^B$.

In the appendix we prove that given a min-cost flow f the weights of the bipartite-like graph edges $w_f(u \overset{\min}{\to} v)$ are computed in $O(1)$ given the flow in area-j nodes (layer-3 nodes in G_f^8) f^j, the flow in type-i (layer-6) nodes f^i, and the flow in area-j type-i (layer-4) nodes f_j^i. Thus BGA will hold these values and will update the edge weights of G_f^B according to them.

BGA imitates the behavior of SSP where each iteration in BGA corresponds to the equivalent iteration in SSP. We now present a sketch of BGA: We first initiate the zero flow value $f_i = f^j = f_i^j = 0$ and define the node potentials π on G_f^B using Bellman-Ford similarly to the initial step of SSP. Then, in every iteration, BGA will do the following steps: 1) Find the shortest $x - y$ path in G_f^B using Dijkstra (which has the same weight as the shortest path SSP founds). 2) Update the node potentials π and flow values f_i^j, f_i and f^j values as described in the appendix. Note that the components of the optimal placement correspond to the optimal flow $L_i^j = f_i^j$ are updated as well . 3) Update the weight of the graph edges of G_f^B according the new flow values and node potentials function.

The optimality of BGA is proved by the following theorem:

Theorem 7.2: At the end of the n^{th} iteration of BGA, whose placement components L_i^j equal to the flow in area-j type-i nodes f_i^j (i.e $L_i^j = f_i^j$), has a minimal placement cost $C_p(L)$ among all placements with number of resources less than or equal to n i.e $|L| \doteq \sum_{i=1}^m \sum_{j=1}^k \leq n$.

The complexity of BGA is shown by the following theorem:

Theorem 7.3: In every iteration BGA computes Dijksra's algorithm over a graph with $O(m+k)$ vertices and $O(mk)$ edges takes $O(mk)$. Updating bipartite edge weights is linear by the size of the path, and is bounded by the number of vertices $O(m+k)$. Thus, the complexity in every BGA iteration is at most $O(mk)$.

Remark 7.4: In a similar manner, we can construct a similar bipartite-graph (with different costs) that solves the bounded problem with a complexity of $O(smk)$, where $s = \sum_{j=1}^k s^j$ is the total storage. This algorithm is superior to the bipartite-graph algorithm introduced in [3] (whose complexity was $O(skm(k+m))$ and was limited for the basic bounded model). More details can be found in the appendix.

B. The Non-Negative Weight Stopping Rule and the SSP convexity Theorem

Next, we introduce the non-negative weight criterion, a stopping rule which must be used to bound the number of iterations. The non-negative weight criterion checks in every iteration of BGA if the shortest path weight $w^{\pi}_{f}(p_{opt})$ (which by Theorem 7.1 equals to the shortest path weight of SSP) is non-negative ($w^{\pi}_{f}(p_{opt}) \geq 0$). If it is non-negative then BGA terminates, and the placement associated with the flow, i.e $L^{j}_{i} = f^{j}_{i}$, solves the unbounded placement problem.

This is a non-obvious stopping rule: One must show that when BGA stops when it reaches an optimal solution. That is, when the shortest path weight is $w^{\pi}(p_{opt}) \geq 0$ then the placement $L^{j}_{i} = f^{j}_{i}$ is an optimal placement among **all** placements.

To prove the optimally of non-negative rule we prove the SSP convexity theorem. That means that the weights of the shortest paths, which are the marginal costs of the min cost flow weight $w(f)$, are monotonically non-decreasing.

Theorem 7.5 (The SSP Covexity theorem): Let G be a general graph with no negative cycles with respect to a weight function w. Let p_i be the shortest path in the i^{th} iteration of SSP. Then, weights of the shortest paths are monotonically non-decreasing i.e $w(p_i) \leq w(p_{i+1})$.

Theorem 7.5 holds even on the 8-layer graph G^8, which has infinitely many nodes. The theorem follows by proving that the shortest path weight between the source x and a vertex v is monotonically increasing.

Proof of Theorem 7.5: Denote by $d_f(u, v)$ the shortest path between u and v in the residual graph G_f respect to w_f. By the definition of the weight function we have $d_{f_i}(x, y) = w_{f_i}(p_i)$ in every i iteration of the SSP algorithm. Also, by Theorem 6.2 there exists a node potentials function π_{i+1} such that $w^{\pi_{i+1}}(e) \geq 0$ for every edge e in $G_{f_{i+1}}$.

Suppose by contradiction that in the i-iteration (of SSP) the weights of shortest paths are decreasing, i.e $d_{f_i}(x, y) = w_{f_i}(p_i) > w_{f_{i+1}}(p_{i+1}) = d_{f_{i+1}}(x, y)$. Also, for every path p between a and b the weight of p respect to the reduce weights $w^{\pi_{i+1}}$ is $w^{\pi}(p) = w(p) - \pi_{i+1}(a) + \pi_{i+1}(b)$ (a proof can be seen in [18]). Thus, we can assume the weights of shortest paths **respect to the reduce weights** $w^{\pi_{i+1}}$ are decreasing, i.e $d^{\pi_{i+1}}_{f_i}(x, y) > d^{\pi_{i+1}}_{f_{i+1}}(x, y)$. We will choose a vertex v such that $d^{\pi_{i+1}}_{f_{i+1}}(x, v)$ is minimal among all vertices with decreasing shortest path, i.e., $d^{\pi_{i+1}}_{f_i}(x, v) > d^{\pi_{i+1}}_{f_{i+1}}(x, v)$.

The vertex v cannot be the source vertex x, otherwise $d^{\pi_{i+1}}_{f_i}(x, v) = d^{\pi_{i+1}}_{f_{i+1}}(x, v) = \pi_{i+1}(x) - \pi_{i+1}(v)$. Thus, there is a vertex u such that the shortest path between x and v in $G_{f_{i+1}}$ ends with (u, v). WLOG, the optimal path between x and u does not decrease. Thus, we can imply that

$$d^{\pi_{i+1}}_{f_{i+1}}(x, v) = d^{\pi_{i+1}}_{f_{i+1}}(x, u) + w^{\pi}_{f_{i+1}}(u, v) \geq d^{\pi_{i+1}}_{f_i}(x, u) + w^{\pi_{i+1}}_{f_{i+1}}(u, v). \tag{9}$$

If $w^{\pi_{i+1}}_{f_{i+1}}(u, v) \geq w^{\pi_{i+1}}_{f_i}(u, v)$, then by the definition of a shortest path we have

$$d^{\pi_{i+1}}_{f_i}(x,u) + w^{\pi_{i+1}}_{f_{i+1}}(u,v) \geq d^{\pi_{i+1}}_{f_i}(x,u) + w^{\pi_{i+1}}_{f_i}(x,u) \geq d^{\pi_{i+1}}_{f_i}(x,v). \tag{10}$$

Combining Eq (9) and Eq (10) will imply that the shortest path to v is not a decreasing - a contradiction. Thus, the reduced weights obey $w^{\pi_{i+1}}_{f_{i+1}}(u,v) < w^{\pi_{i+1}}_{f_i}(u,v)$ and therefore the original weights in G_f obey the same condition, i.e, $w_{f_{i+1}}(u,v) < w_{f_i}(u,v)$. By the definition of the residual graph, $w_{f_{i+1}}(u,v) < w_{f_i}(u,v)$ can be only if $w_{f_i}(u,v) = \infty$, where its residual capacity of the edge is $c_{f_i}(u,v) = 0$. Thus, the flow in edge (u,v) equals to its positive capacity i.e, $f(u,v) = cu, v > 0$, and the residual capacity of its reverse edge is $c_{f_i}(v,u) = -f(u,v) < 0$. Also, the weight of the reverse edge equals to the additive inverse of its original edge $w_{f_i}(v,u) = -w_{f_{i+1}}(u,v)$ and therefore $w^{\pi_{i+1}}_{f_i}(v,u) = -w^{\pi_{i+1}}_{f_{i+1}}(u,v) < 0$.

The shortest path p_i in the i-iteration cannot pass through (u,v) (which has infinite weight), otherwise SSP will terminate. If p_i does not pass through (v,u), then the edge weight of (u,v) in the iteration, i.e. $w_{f_{i+1}}(u,v)$, is infinite, and therefore $\infty = w_{f_{i+1}}(u,v) = w_{f_i}(u,v)$- a contradiction to $w^{\pi_{i+1}}_{f_{i+1}}(u,v) < w^{\pi_{i+1}}_{f_i}(u,v)$. Thus, p_i, which is a shortest path, must pass through (v,u), and therefore

$$d^{\pi}_{f_i}(x,u) = d^{\pi}_{f_i}(x,v) + w^{\pi}_{f_i}(v,u). \tag{11}$$

But from Eq (9) the definition of v we obtain

$$d^{\pi}_{f_i}(x,v) > d^{\pi}_{f_{i+1}}(x,v) \geq d^{\pi}_{f_i}(x,u) + w^{\pi}_{f_{i+1}}(u,v) = d^{\pi}_{f_i}(x,u) - w^{\pi}_{f_i}(v,u) = d^{\pi}_{f_i}(x,v).$$

A contradiction.

■

Theorem 7.1 and Theorem 7.5 lead directly to the optimality of the stopping rule:

Corollary 7.6: Let f_i be the flow that BGA finds at the i^{th} iteration. Let p_i be the shortest path in the i^{th} iteration of BGA. Assume that the shortest path p_{i_0} is the first non-negative path found by BGA, i.e $w(p_{i_0-1}) < 0 \leq w(p_{i_0})$. Then f_{i_0} has minimum weight over all possible min-cost flows (i.e $f_{i_0} = \arg\min_{\text{f is min-cost flow}} w(f)$). Thus, the placement L associated with the flow f_{i_0} solves the unbounded problem.

Finally, let $|L| = \sum_{i=1}^{m} \sum_{j=1}^{k} L_i^j$ denote the number of resources in placement L. In the appendix (Section A) we prove that in every iteration of BGA the number of resources increases exactly by one resource. Thus, the number of iterations that BGA uses is $|L_{opt}|$, where L_{opt} is the optimal placement for the unbounded problem (we assume in Section III that an optimal placement exists). The complexity of the BGA is $O(|L_{opt}|km)$ where k is number of areas and m is number of resource types.

VIII. Dynamic problem: sensitivity analysis and a solution

In a dynamic environment one has to deal with the time varying demands $D(t)$ and with the need to recompute the optimal placement and reposition the resources as the demand changes. We start our analysis by showing that any algorithm **A** that insists on recomputing the optimal placement and reposition the resources accordingly at every time t may be subject to a very high repositioning cost $r(A)$ (see definition in Subsection III-D). This holds even if the prior placement computed by A, $L_A(t-1)$, is optimal with respect to $D(t-1)$ (i.e $L_A(t-1) = L_{opt}(D(t-1))$) and the demands sets $D(t-1), D(t)$ are extremely close to each other (which we denote **strongly ϵ-near**). A high reposition cost may be quite undesired since it may imply that the system operations must be held due to lengthy repositioning.

To address this issue, we provide an online algorithm called **Lazy ONline Algorithm** (LONA) and denoted as A_{LONA}, which provides a tradeoff between the repositioning cost and the placement deviation. We prove that LONA satisfies the following conditions: 1) Its placement deviation cost (defined in Subsection III-D) $C_{dev}(A_{LONA})$ is bounded by a given threshold ϵ, and 2) Its repositioning cost stays 0, if the demand set $D(t)$ is close to the previous used set (which we called **weakly ϵ-near**).

A. Sensitivity analysis

We start with the definition of strongly ϵ-near.

Definition 3: Given two vectors $\hat{v} = (v_1, \dots v_n)$ and $\hat{u} = (u_1, \dots u_n)$, the **$L_1$-distance** between them is $d(\hat{v}, \hat{u}) = \sum_{i=1}^{n} |v_i - u_i|$. Given two discrete distributions N_1, N_2 defined over the same support set $\{0, 1, 2 \dots\}$, we define the **L_1-CDF distance** as the L_1-distance between the CDF vectors of N_1 and N_2 i.e $d(N_1, N_2) = \sum_{n=0}^{\infty} |\Pr(N_1 \leq n) - \Pr(N_2 \leq n)|$. The demand sets $D = \{D_i^j\}$, $D' = \{D'^j_i\}$ are called **strongly ϵ-near** iff the following conditions hold: 1) $D_i^j = D'^j_i$ for all $(i, j) \neq (i_0, j_0)$ and 2) $d(D_{i_0}^{j_0}, D'^{j_0}_{i_0}) < \epsilon$.

In the following theorem we show that given strongly ϵ-near demand sets $D(t), D(t-1)$, and an optimal placement $L_{opt}(D(t-1))$, computing the optimal placement for the next time slot $L_{opt}(D(t))$ yields a high repositioning cost. Therefore, every online algorithm A which computes the optimal placement in successive time slots $t-1, t$ has a high repositioning cost:

Claim 8.1: For every $\epsilon > 0$ there exist two demand sets $D(t-1), D(t)$ which are strongly ϵ-near while the L_1-distance between the optimal placements $L_{opt}(D(t-1))$ and $L_{opt}(D(t))$, is larger than the maximum storage value i.e $d(L_{opt}(D(t-1)), L_{opt}(D(t))) \geq \max_j s^j$.

A proof is given in the appendix.

B. The Lazy ONline Algorithm - LONA

To introduce LONA we present the following definitions:

Definition 4: Let $D = \{D_i^j\}$, $D' = \{D'^j_i\}$ be two demand sets. The **demand distance** between D and D', denoted as $d(D, D')$ is

$$d(D, D') = \sum_{i=1}^{m} \sum_{j=1}^{k} d(D_i^j, D'^j_i) R_i^{loc} + \sum_{i=1}^{m} d(D_i, D'_i) R_i^{glo}, \tag{12}$$

where R^{loc} and R^{glo} are respectively the local and global differential revenue constants. This equation resembles Eq (5). Demands sets D and D' are called **weakly ϵ-near** if the distance between them is less than ϵ, i.e $d(D, D') < \epsilon$.

LONA is a simple lazy algorithm with a threshold parameter ϵ. At time t LONA holds a reference demand set $D_{ref}(t)$ equaling $D(\tau)$ for some $\tau < t$, where τ is the last time where the algorithm modified its placement. It also holds as reference the optimal placement $L_{opt}(D(\tau))$. The operation of LONA at t is simple: It compares $D_{ref}(t)$ with $D(t)$ and checks whether they are weakly ϵ-near; if they are, then the output of LONA, $L_A(t)$, is identical to $L_A(t-1)$ and equals to $L_{opt}(D(\tau))$; otherwise LONA computes the optimal placement $L_{opt}(D(t))$ and sets this is as its output $L_A(t)$. In this latter case LONA also updates the reference demand set $D_{ref}(t)$ to be $D(t)$ and the reference optimal placement to $L_{opt}(D(t))$. The optimal placement $L_{opt}(D(t))$ can be calculated by an optimal placement algorithm (which can be either of SSP or BGA offline algorithms, or the online out-of-kilter algorithm presented in the appendix.

We prove that LONA has a low total cost deviation, even in the cases where it does not recompute the optimal placement:

Theorem 8.2: Let ϵ be the threshold parameter of LONA. If the demand reference $D_{ref}(t)$ and the demand $D(t)$ are weakly ϵ-near, then the deviation cost respect to $D(t)$ is bounded by ϵ, i.e, $C_{dev}(A_{LONA}, D(t)) = |C_p^{D(t)}(L_A(t)) - C_p^{D(t)}(L_{opt}(D(t)))| < \epsilon$.

The theorem is proved by examining the optimal flow problem, tracking the out-of-kilter algorithm [18] which finds the min-cost flow f_{opt} from a non-optimal flow f, and showing that it changes the flow weight by at most ϵ using Theorem 6.2.

By Theorem 8.2 we have the following corollary:

Corollary 8.3: LONA satisfies the following properties: 1) Its placement deviation cost is less than the threshold parameter ϵ, i.e $C_{dev}(A_{LONA}) < \epsilon$, and 2) its repositioning cost is 0 if the demand sets are weakly ϵ-near.

IX. Capacity Allocation - Performance evaluation

In this section, we evaluate the unbounded placement solution to study the rate of approximation as a function of the number of resources. [3] We consider a demand that follows a Zipf distribution (which is consistent with prior analytical works [12], [19] and VoD empirical results [20], [21]). This means that there exists a real number $e > 0$ such that the probability for a single request to demand a type-i resource is $p_i = \frac{1}{i^e \cdot H}$ for all resource types

[3]Comparison to non-optimal solutions, e.g proportional mean ([12]), was carried in [3].

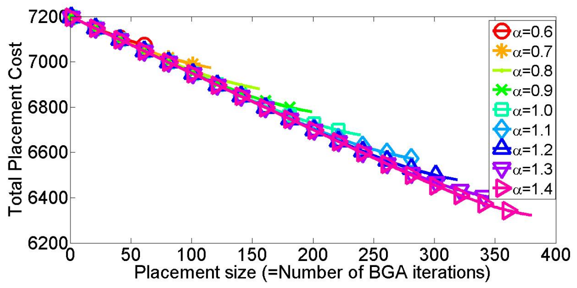

Fig. 4. Total cost of placement as a function of the number of resources (# iterations in BGA)

$1 \le i \le m$, where $H = \sum_{j=1}^{m} \frac{1}{j^e}$. The demand in area j is assumed to be a Poisson distribution of parameter λ^j and the demand for type-i resources originating from area j, D_i^j, is a Poisson distribution with parameter of $p_i \cdot \lambda^j$.

In Fig. 4 we depict the total cost of the optimal placement as a function of its size. As it is unclear how future demands will behave (prior references mentioned above used a Zipf parameter of values 0.56 - 1.5, depending on the study) we consider a wide range of Zipf distributions and vary the Zipf parameter from 0.6 to 1.4. We set receptively the number of resources types to be $m = 100$, the number of areas $k = 3$, the service cost parameters $C_{unsat} = 3 > C_{rem} = 2 > C_{loc} = 0$, and the demand distribution parameters $\lambda^1 = 400 < \lambda^2 = 800 < \lambda^3 = 1200$. In addition to service cost, we incorporate a linear resource cost (See Eq. (1)) where we set $p_i^j = h^j = r_i = 0.5$. We assume that every resource type can grant up to $B_i = 5$ servers.

Fig. 4 reveals several important properties regarding the behavior the optimal placement: 1) As the Zipf parameter increases, i.e. the demand for esoteric resources decreases, the placement size increases. 2) The number of BGA iterations equals to the placement size (See section VII-A). Thus, as the demand for esoteric resources decreases the number of iterations for convergence of BGA increases. 3) Before reaching its final iterations, the cost derived by BGA unbounded decreases almost linearly. This means that using another stopping rule for BGA unbounded, such as stopping whenever the marginal cost decreases below some positive threshold, is inefficient and will degrade the performance of the algorithm. In other words, only our presented non-negative weight criterion should be taken as a stopping rule for the BGA unbounded algorithm.

X. Concluding Remarks

We addressed the problem of network resource placement. We introduced a modeling framework that captures a wide variety of design parameters and problems, accounting for stochastic demands. We proposed an algorithmic methodology that overcomes the dimensionality and efficiently solves a variety of problems, including static and time varying placements as well as capacity planning. The methodology applies to a wide variety of practical applications.

References

[1] "Amazon EC2 home page. http://aws.amazon.com/ec2," 2013.

[2] "Microsoft Azure home page. http://www.windowsazure.com," 2013.

[3] Y. Rochman, H. Levy, and E. Brosh, "Resource placement and assignment in distributed network topologies," in *IEEE INFOCOM*, Turin, Italy, April 2013.

[4] Y. Li, S. Ranka, and S. Sahni, "In-advance path reservation for file transfers in e-science applications." *The Journal of Supercomputing*, vol. 59, no. 3, pp. 1167–1187, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/tjs/tjs59.html#LiRS12

[5] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes, "Globally-optimal greedy algorithms for tracking a variable number of objects," in *CVPR*. IEEE, 2011, pp. 1201–1208.

[6] Z. Drezner and H. W. Hamacher, *Facility Location: Applications and Theory*. Springer, 2002.

[7] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *IEEE INFOCOM*, Anchorage, AK, USA, April 2001.

[8] F. L. Presti, C. Petrioli, and C. Vicari, "Distributed dynamic replica placement and request redirection in content delivery networks," in *MASCOTS*, 2007, pp. 366–373.

[9] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE Journal on Selected Areas in Communications*, vol. 99, p. pp, 2013.

[10] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. Lau, "Scaling Social Media Applications into Geo-Distributed Clouds," in *IEEE INFOCOM*, Orlando, Florida, USA, March 2012.

[11] H. Xu and B. Li, "Joint Request Mapping and Response Routing for Geo-distributed Cloud Services,," in *IEEE INFOCOM*, Turin, Italy, April 2013.

[12] S. Tewari and L. Kleinrock, "Proportional replication in peer-to-peer networks," in *IEEE INFOCOM*, Barcelona, Spain, April 2006.

[13] Y. P. Zhou, T. Z. J. Fu, and D. M. Chiu, "Statistical modeling and analysis of p2p replication to support vod service," in *IEEE INFOCOM*, Orlando, FL , USA, July 2011.

[14] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale vod system," in *ACM CoNEXT*, Philadelphia, USA, Dec 2010.

[15] M. Leconte, M. Lelarge, and L. Massoulié, "Bipartite graph structures for efficient balancing of heterogeneous loads," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS, 2012, pp. 41–52.

[16] R. G. Busacker and P. J. Gowen, "A procedure for determining a family of minimal cost network flow patterns," Operational Research Office, Johns Hopkins University, Baltimore, MD, ORO Technical Report 15, September 1961.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[18] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[19] B. Tan and L. Massoulié, "Optimal content placement for peer-to-peer video-on-demand systems," in *IEEE INFOCOM*, Orlando, FL , USA, July 2011.

[20] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *ACM Internet Measurement Conference*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298310

[21] V. Valancius, N. Laoutaris, L. Massoulie, C. Diot, and P. Rodriguez, "Greening the Internet with Nano Data Centers," in *ACM CoNext*, Rome, Italy, Dec 2009.

[22] Y. Rochman, H. Levy, and E. Brosh, "Max percentile replication for optimal performance in multi-regional p2p vod systems," in *Proceeding of the 9th International Conference on Quantitative Evaluation of SysTems (QEST) 2012*, London, UK, September 2012.

Appendix

Proof of claims

Proof of Claim 4.1: Consider $\{d_i^j\}$ a demand realization, and M be an assignment between the demand and a placement L. We define the *matching cost* of M simply as

$$C(L, d_i^j, M) = \sum_{i=1}^{m} C_i^{loc} \cdot g_i^{loc}(M) + C_i^{rem} \cdot g_i^{rem}(M) + C_i^{unsat} \cdot g_i^{unsat}(M) \tag{13}$$

where $g_i^{loc}(M)$, $g_i^{rem}(M)$ and $g_i^{unsat}(M)$ are respectively the corresponding number of type-i requests granted by M from a local area, granted from a remote area, and unsatisfied ones. Similarly to the revenue case, we define the minimal service cost between placement L and realization $\{d_i^j\}$ as the optimal assignment minimizing the matching cost i.e $C(L, d_i^j) = \arg\min_M C(L, d_i^j, M)$.

We will prove that the *total sum*, which simply as the sum of the cost matching and the revenue matching $R(L, d_i^j, M) + C(L, d_i^j, M)$ is constant. Consider the marginal value of a type-i request req_i in the total sum according to the following cases:

1) If req_i is granted locally, then it is also granted globally. Thus its marginal value of the matching revenue is $R_i^{loc} + R_i^{glo} = C_i^{unsat} - C_i^{loc}$, and the marginal value in the total sum equals to C_i^{unsat}.
2) If req_i is granted remotely, then it is granted globally but not locally. Thus its marginal value of the matching revenue $R_i^{glo} = C_i^{unsat} - C_i^{rem}$ and in the total sum C_i^{unsat}.
3) If req_i is an unsatisfied request, then it nor granted globally or locally. Thus, its marginal value of the matching revenue is 0 and the marginal revenue in the total sum is C_i^{unsat}.

Thus every type-i request has a marginal value of C_i^{unsat} to the total sum. This implies that total sum equals to $C(L, d_i^j, M) + R(L, d_i^j, M) = \sum_{i=1}^{m} C_i^{unsat} D_i$., where D_i is the number of type-i requests. Note that $\sum_{i=1}^{m} C_i^{unsat} D_i$ is independent of the matching M. Thus, a matching M_0 that maximizes matching revenue (i.e $R(L, d_i^j, M_0) = R(L, d_i^j)$) must minimize the matching cost (i.e $C(L, d_i^j, M_0) = C(L, d_i^j)$. Therefore,

$$C(L, d_i^j) + R(L, d_i^j) = C(L, d_i^j, M_0) + R(L, d_i^j, M_0) = \sum_{i=1}^{m} C_i^{unsat} D_i. \tag{14}$$

Taking expectation over the demand set D yields the theorem statement. ■

Proof of Claim 4.3: We assume in Section III that the resource cost $C_r(L)$ is semi-separable and by Eq (4) the service revenue $E(R(L))|_D$ is a semi-separable function. Thus, subtracting the resource cost with service revenue, which equals to the loss function (Eq (3)), is also a semi-separable function.

We assume in Section III that total placement cost $C_p(L)$ is convex. Also, by claim 4.1 loss function equals to the sum of the placement cost and a constant $c = \sum_i E(D_i) C_i^{unsat}$, i.e $loss_p(L) = C_p(L) + c$. Thus, the loss function is also convex. ■

Proof of Claim 5.1: It is sufficient to prove that for every constant C non-negative integer valued random variable X we have $E(\min(X,C)) - E(\min(X,C-1)) = \Pr(X \geq C)$. A proof of it can be found in [22] Claim 6.4. ■

Proof of Lemma 5.3: We will prove that for every placement $P = \{P_i^j\}$ its loss $loss_p(P)$ is not larger than the optimal flow weight f_{opt}, which equals to the loss of its correspond placement L.

Let P be a placement. We define a flow $f(P)$ as follows:

- The disjoint paths x-(a^j,n)-a^j, (a^j,t_i)-(a^j,t_i,n)-t_i, t_i-(t_i,n)-y are assigned respectively with one unit of flow if $P^j \geq n$, $P_i^j \geq n$ and $P_i \geq n$; Otherwise, the correspond path is assigned with zero unit of flow.
- The flow between (a^j,t_i), and (a^j,t_i) is P_i^j
- The flow between x and y is $s - \sum_{i=1}^{m}\sum_{j=1}^{k} P_i^j$

One can verified that 1) f preserves the capacity constraint and conservation of flows 2) the flow of $f(P)$ equals to $|f(P)| = s$ and 3) the weight of $f(P)$ equals to loss cost of P, i.e $w(f(P)) = loss_p(P)$. Since f_{opt} is a min-cost flow of flow value s, then $loss_p(P) = w(f(P)) \leq w(f_{opt})$ as required.

As the marginal-differential functions $\Delta\gamma$ are monotonically increasing, WLOG min-cost flow f_{opt} assigns one unit of flow respectively to x-(a^j,n)-a^j, (a^j,t_i)-(a^j,t_i,n)-(t_i),(t_i)-(t_i,n)-y if $L^j \geq n$, $L_i^j \geq n$ and $L_i \geq n$. Therefore the weight of f_{opt} equals to the loss of L. ■

Proof of Theorem 6.2: We will prove the claim by induction on the iteration number k. In the initial step ($k=0$) of the SSP algorithm, all edges (v_1,v_2) in G have reduced weight equals to $w_f^{\pi}(v_1,v_2) = w(v_1,v_2) + d(v_1) - d(v_2)$. The shortest path between s and v_2 is not strictly longer than the shortest path between s and v_1 concatenated with the edge (v_1,v_2), i.e $w(v_1,v_2) + d(v_1) \geq d(v_2)$. Therefore, $w_f^{\pi}(v_1,v_2) \geq 0$. Every other edge e in the residual graph G_f which is not in the original graph G must have infinity reduced flow (i.e. $w_f^{\pi}(v_1,v_2) = \infty$). Thus in the initial step of our version to the SSP algorithm, π satisfies the non-negative reduced weight condition.

Denote the node potentials function in the k iteration as π_{k-1}. Then at [18] they prove that if π_{k-1} satisfies the non-negative reduced weight condition, then so as π_k. Thus, the claim is proved, as required. ■

Proof of Theorem 7.1: Let $p_{opt} = (x, v_1, v_2, \ldots, v_s, y)$ be a shortest path in G_f^8. We denote by $u \to v$ a monotone path that begins in a vertex u and ends in vertex v. We will proved that there exist area nodes $a(1), a(2), \ldots, a(n)$ and a type nodes $t(1), t(2), \ldots, t(n)$ such that $p_{opt} = x \to a(1) \to t(1) \to a(2) \to \ldots \to t(n) \to y$. WLOG, the optimal path p_{opt} is not the edge (x,y).

By Theorem 6.2 all edges in G_f^8 have non-negative reduced weights. Thus, the graph does not contain negative cycles (a proof can be seen in [18]). WLOG the optimal path p_{opt} does not contain cycles, otherwise the non-negative cycles can be omitted from the path.

The first node in the optimal path v_1 must be a layer-2 node $(a(1), num)$ in G_f^8. The second node, v_2 must be an area node $a(1)$; Otherwise, v_2 will be the source node x and the path p_{opt} will contains a cycle. Thus, the path p_{opt} begins with a forward path between the source node x and an area node $a(1)$ (i.e $v_1 = (a(1), num)$ and $v_2 = a(1)$). In the same way one can show that p_{opt} ends with a forward path between a type node $t(n)$ and the sink node y (i.e. end with the path $t(n)$-$(t(n), num)$-y. Similarly, one can show that by induction that the path between $a(1)$ and $t(n)$ alternately passing between forward paths between area nodes $a(i)$ and type nodes $t(i)$ and backward path between type node $t(i)$ and area node $a(i+1)$. Thus the optimal path is composed from monotone paths.

WLOG, every monotone path in p_{opt} is a shortest monotone path i.e

$$p_{opt} = x \overset{\min}{\to} a^{j_1} \overset{\min}{\to} t^{i_1} \overset{\min}{\to} a^{j_2} \ldots \overset{\min}{\to} a^{j_e} \overset{\min}{\to} t^{i_e} \overset{\min}{\to} y,$$

Therefore, the weight of optimal path in G^8_f, p_{opt}, equals to the weight of a path p^B in G^B_f ($w_f(p_{opt}) = w_f(p^B_{opt})$). Also, every path p^B in G^B has an equivalent path p in G^8_f with the same weight (i.e $w_f(p) = w_f(p^B)$). Thus, the weight of p_{opt} equals to the weight of the shortest path in G^B_f as required. ■

Proof of Theorem 7.2: This simply follows from Lemmas A.3, A.2 and Theorem 7.1. ■

Proof of Theorem 7.2: This simply follows from Lemmas A.3, A.2 and Theorem 7.1. ■

Proof of Corollary 7.6: This can be prove by the fact that the flow weight in the i^{th} iteration equals to $w(f_i) = w_{f_{i-1}}(p_{i-1}) + w(f_{i-1})$. ■

Proof of Claim 8.1: We set $D(t)$ to the instance where the number of requests is zero, i.e. $\Pr(D(t)^j_i = 0) = 1$ for all resource type i and area j. Placing in every region only type-2 resources is optimal (i.e. $L_{opt}(D(t-1))^j_2 = s^j$ for every j).

The demand set in the next time slot $D = D(t+1)$ is defined as follows: 1) The probability that the number of requests for resource of type-1 in area 1 is more than n with probability of $\frac{\epsilon}{2^{n+1}}$ (i.e $\Pr(D^1_1(t) \geq n) = \frac{\epsilon}{2^{n+1}}$). 2) The number of requests for type-i resources in area j, such that $(i,j) \neq (1,1)$ is 0 (i.e $\Pr(D(t)^j_i = 0) = 1$). An optimal placement for $D(t+1)$ is the placement allocating to every region a type-1 resource. Note that $D(t)$ and $D(t+1)$ are strongly ϵ-near, and the distance between $L_{opt}(D(t-1))$ $L_{opt}(D(t))$ is s, as required. ■

Proof of theorem 8.2: To prove the theorem, we will use the out-of-kilter algorithm (presented in Section A) that solves the min-cost flow problem. Suppose the demand set $D = D(\tau) = D_{ref}(t)$ was updated to a new demand set $D(t)$. Then the weight function $w()$ which is correspond 8-layer graph G^8_f of the optimal flow $f = f_{opt}$ and node potentials π, was updated a new weight function w' (see Fig. 2). The reduce weights condition correspond to the original optimal flow f (See Section VI) does not takes place as the weight function changed. Thus, we use the out-of-kilter algorithm that given the (old) node potentials π and (old) optimal flow f finds a new min-cost optimal flow f' with new node potentials π', respect to the new demand $D' = D(t)$. We will prove that flow weight is changed by at most ϵ (i.e. $|w'(f) - w'(f')| < \epsilon$), and therefore, by Lemma 5.3, the theorem is be proved.

Let $e = (u, v)$ be an out-of-kilter edge in the 8-layer graph G^8_f respect to the initial node potentials reduce weight w'^{π}, i,e., $0 > w'^{\pi}_f(e)$. On the other hand the reduce weight respect to the previous weight, w^{π}_f, is non-negative and therefore $w^{\pi}_f(e) \geq 0$. By the definition of reduce weights formula of w^{π} and w'^{π} we obtain the following formula

$$w(u, v) - w'(u, v) \geq -w'^{\pi}_f(u, v) = -w'^{\pi}(u, v) > 0, \tag{15}$$

and particularly we have $w(u, v) \neq w'(u, v)$.

For the sake of the proof we call respectively to the edges entering nodes (a^j, n), (a^j, t_i, n), (t_i, n) *area edges*, *area+type edges* and *type edges*. The weight of other edges is zero weight (i.e. $w(e) = w'(e) = 0$), according to Fig. 2. The weight of area edges equals to $w(e) = w'(e) = \Delta\zeta^j(n) = \zeta^j(n) - \zeta^j(n-1)$ (according to Claim 5.1) which is not dependent on the demand distribution D. Thus, out-of-kilter edges must be area+type edges and the type edges.

Denote cyc_i the cycle found in i^{th} iteration in State 7 of the out-of-kilter Algorithm. Then augmenting the flow along cyc_i increases (State 9) the flow weight $w'(f)$ by $-w'(cyc)$, where $w'(cyc) = \sum_{e\in cyc} w'(e)$. Thus, if the out-of-kilter algorithm runs over t iterations, then

$$w'(f) - w'(f') = -\sum_{i=1}^{t} w'(cyc_i) \tag{16}$$

Denote π_i the node potentials in the i^{th} iteration. Then, the weight of every cycle C equals to the cycle weight respect to node potentials π_i, i.e $w'(C) = w'^{\pi_i}(C) = \sum_{e\in C} w'^{\pi_i}(e)$. Let A_i denote the set of out-of-kilter edges after the i^{th} iteration. Then according to Lemma A.5 every in-kilter edge $e \in cyc_i \bigcap (A_i)^c$ has reduced weight of $w'^{\pi_i}(e) = 0$. Thus, if we denote by $B_i = cyc_i \bigcap A_i$ the out-of-kilter edges in the i^{th} cycle, then the weight of every cycle cyc_i equals to the sum of out-of-kilter edges i.e. $w'(cyc_i) = \sum_{e\in B_i} w'^{\pi_i}(e)$. Thus, if we denote by $B = \bigcup i = 1^t B_i$ the out-of-kilter edges then we yield that $w'(f) - w'(f') = \sum_{e\in B} w'^{\pi_i}(e)$.

The area+type and type edges have capacity of 1. Thus, all out-of-kilter edges have a kilter number, which is the residual capacity $c_f(e)$, of 1. If $e \in B_i$ is an out-of-kilter edge then after augmenting through cyc_i its residual capacity decreases and it becomes an in-kilter edge. Thus, the sets B_i for are disjoint in pairs i.e, $B_i \cap B_j = \emptyset$ for $i \neq j$, and therefore

$$w'(f) - w'(f') = -\sum_{i=1}^{t} \sum_{e\in B_i} w'^{\pi_i}(e). \tag{17}$$

Let $(u, v) = e \in B_i$ be an out-of-kilter edge. Then e is an out-of-kilter edge in the $i-1$-iteration with a negative reduce weight i.e. $w'^{\pi_{i-1}}(e) < 0$. The node potentials of i iteration equals to $\pi_i = \pi_{i-1} - d$, and therefore reduce weight of edge (u, v) is $w'^{\pi_i}(e) = w'^{\pi_{i-1}}(e) + d(u) - d(v)$.

But the weight in the i iteration of edge e (Step 3) equals to $\max(0, w'^{\pi_{i-1}}(e)) = 0$, and the shortest path to v

is not longer than the shortest path to u, i.e. $d(u) \leq d(v)$. Therefore, we imply that $w'^{\pi_i}(e) \geq w'^{\pi_{i-1}}(e)$ for every out-of-kilter edge e, and moreover, one can imply by induction that $w'^{\pi_i}(e) \geq w'^{\pi_0}(e)$, where $\pi_0 = \pi$ is the node potentials in the initial iteration. Since $e \in B_i$ is an out-of-kilter in the initial iteration ($e \in A_0$) by Eq (15) we obtain $w'^{\pi_i}(e) \geq w'^{\pi_0}(e) \geq -|w'(e) - w(e)|$, and by Eq (17) we imply that.

$$w'(f) - w'(f') \leq \sum_{e \in B} |w'(e) - w(e)|. \tag{18}$$

According to Claim 5.1, if $e \in B$ is an area+type edge enters to vertex (a^j, t_i, n), then

$$w(e) - w'(e) = R_i^{loc}[\Pr(D_i^j \geq n \cdot B_i) - \Pr({D'}_i^j \geq n \cdot B_i)]. \tag{19}$$

Similarly, if $e \in B$ is a type edge enters to vertex (t_i, n) then

$$w(e) - w'(e) = R_i^{glo}[\Pr(D_i \geq n \cdot B_i) - \Pr({D'}_i \geq n \cdot B_i)] \tag{20}$$

Then combining Eq (19), (20), (18) with the definitions of the demand distance (Eq (12)) and weakly ϵ-near yields that

$$|w'(f) - w'(f')| \leq d(D, D') < \epsilon$$

As required.

■

THE ASSIGNMENT ALGORITHM

To present the assignment algorithm we call a type-i resource *partial-loaded* if the number of requests assigned to resource is strictly less than its capability B_i. If the number of requests assigned to resource equals to its capability, then the resource is called *fully-loaded*.

In Algorithm 1 we derive a specific matching algorithm tailored for maximizing the revenue matching.

Algorithm 1 The assignment algorithm

```
Require: An placement $L = \{L_i^j\}$, the demand realization $d_i^j$, and resource capacities $B_i$.
1: for all movie $i$ do
2:   for all region $j$ do
3:     Assign $\min(B_i \cdot L_i^j, d_i^j)$ type-$i$ requests from area $j$ to $\min(L_i^j, \left\lceil \frac{d_i^j}{B_i} \right\rceil)$ type-$i$ resources in area-$j$.
4:   end for
5:   while There is an unmatched type-$i$ request and a type-$i$ partial-loaded resource do
6:     Match the request with the resource.
7:   end while
8: end for
```

By the following claim we prove the assignment matching optimality:

Claim A.1: Given placement $L = \{L_i^j\}$ and demand realization d_i^j, then the following claims holds:

1) For every matching algorithm we have $g_i^{loc}(M) \leq \sum_{j=1}^{k} \min(B_i \cdot L_i^j, d_i^j)$ and $g_i^{glo}(M) \leq \min(L_i \cdot B_i, d_i)$.
2) The assignment algorithm yields a revenue as in Eq (4). Moreover, the assignment algorithm maximizes the matching revenue $R(L, d_i^j, M)$.

Proof:

Proof of part 1): Let M be some matching algorithm, and denote $g_{i,j}^{loc}(M)$ as the number of type-i granted locally in region j. Then the number granted locally requests is less than the number of requests, i.e $g_{i,j}^{loc}(M) \leq d_i^j$. Also, the number of requests granted locally is less than the number of resources multiple its capability $g_{i,j}^{glo}(M) \leq L_i^j \cdot B_i$. Therefore, we have $g_{i,j}^{loc}(M) \leq \min(L_i^j \cdot B_i, d_i^j)$, and $g_i^{loc}(M) \leq \sum_{j=1}^{k} \min(B_i \cdot L_i^j, d_i^j)$ as required. $g_i^{glo}(M) \leq \min(L_i \cdot B_i, d_i)$ is prove similarly.

Proof of part 2): We denote by M_{opt} the optimal assignment algorithm. In the end of Step 2 the number of type-i requests granted locally is $g_i^{loc}(M_{opt}) = \sum_{j=1}^{k} \min(B_i \cdot L_i^j, d_i^j)$. We will prove $g_i^{glo}(M_{opt}) \geq \min(L_i \cdot B_i, d_i)$. Assume by contradiction otherwise, i.e $g_i^{glo}(M_{opt}) < \min(L_i \cdot B_i, d_i)$. Since $g_i^{glo}(M_{opt}) < d_i$ and $g_i^{glo}(M_{opt}) < L_i \cdot B_i$, there exist a unmatched type-i request and a type-i partial-loaded resource. But in Step 5 we match every unmatched type-i request to matched to a type-i partial-loaded resource - a contradiction. By using part 1) of the claim, we yield that the number of type-i requests granted globally is $g_i^{glo}(M_{opt}) = \min(L_i \cdot B_i, d_i)$. As required.

The algorithm optimality is driven immediately by the first part of the claim.

■

Computation of the weight of edges and node potentials in a bipartite graph

By the following lemmas, we can compute the edges weight and node potentials of the bipartite graph in $O(1)$ (given computing the marginal differential is $O(1)$).

Lemma A.2: Let f be a flow that SSP calculates in its i^{th} iteration in G_f^8. We denote respectively by f_i^j, f^j, f_i the flow through (a^j, t_i), the flow through a^j and the flow through t_i . Then the minimal monotone paths weights in G_f^8 can be computed in $O(1)$ by the marginal-differential functions (Claim 5.1) as given in the following formulas: 1) $w_f(x \overset{\min}{\to} a^j) = \Delta\gamma^j(f^j + 1)$. 2) $w_f(a^j \overset{\min}{\to} t^i) = \Delta\gamma_i^j(f_i^j + 1)$. 3) $w_f(t^j \overset{\min}{\to} a^i) = -\Delta\gamma_i^j(f_i^j)$ if $f_i^j > 0$ and otherwise ∞. 4) $w_f(t_i \overset{\min}{\to} y) = \Delta\gamma_i(f_i + 1)$.

Proof of Lemma A.2: Note that a flow passing through vertex (a^j, n) has a weight of $\Delta\gamma^j(n)$. As the marginal differential functions are monotonically increasing, a min-cost flow must pass through vertices $(a^j, 1), (a^j, 2), \ldots, (a^j, f^j)$. Thus, the weight forward paths x-(a^j, n)-a^j is $\Delta\gamma^j(n)$ iff $n > f^j$ and otherwise ∞. Thus, the minimal monotone path between the source x and an area node a^j must pass through vertices (a^j, f^j+1) with a weight of $\Delta\gamma^j(f^j+1)$. We have prove part 1), and parts 2), 3) and 4) can be proved by a similar way. ■

Lemma A.3: Let f and π be the node-potentials that SSP calculates in its i^{th} iteration in G_f^8, and let $d^B(v)$ be the shortest path weight between x and v in the **bipartite graph** G_f^B (not the 8-layer residual G_f^8 graph) respect to the node potentials π. Then, SSP updates the node potentials as $\pi(v) = \pi(v) - d^B(v)$ for every vertex v in V^B.

Proof of Lemma A.3: For this proof we denote $d^8(v) = d(v)$ as the shortest path weight between x and v in the 8-layer residual G_f^8 graph , with respect to π. Then SSP updates the node potentials as $\pi = \pi - d^8 = \pi - d$ (See Section VI). Thus, it is sufficient to prove that for every vertex v we have $d^B(v) = d^8(v)$.

Let v be a vertex in V^B, and let $p_{opt}^B(v)$ be the shortest path in G^8. Then we have $d^B(v) = w_f^\pi(p_{opt}^B(v))$. In [18] they prove that the reduce weight for every path p between vertices a and b in a general graph G equals to $w^\pi(p) = w(p) - \pi(a) + \pi(b)$ where $w(p)$ is the path weight. Thus, the shortest path respect to the node potentials π equals to $d^B(v) = w_f(p_{opt}^B(v)) - \pi(x) + \pi(v)$

Similarly to the proof Theorem 7.1, there exists a path $p_{opt}^8(v)$ which has the same weight of $p_{opt}^B(v)$ and is the shortest path in G_f^8, namely $w_f(p_{opt}^B(v)) = w_f(p_{opt}^8(v))$. This implies that shortest path respect to π in 8-layer residual equals to $d^8(v) = w_f(p_{opt}^8(v)) - \pi(x) + \pi(v) = w_f(p_{opt}^B(v)) - \pi(x) + \pi(v) = d^B(v)$, as required. ■

BGA COMPLEXITY AND THE BOUNDED CASE

First we prove the BGA complexity according to the following claim.

Claim A.4: BGA runs in $O(lkm)$ where l is the number of iterations

Proof: In every iteration the algorithm finds the shortest paths in G_f^B by Dijkstra algorithm, which costs $O(|E|+|V|\log|V|) = O(km)$. Updating the node potentials and the flow values takes at most $O(|V|) = O(k+m)$ time, and updating the graph weights is at most $O(|E|) = O(km)$ time. ■

To solve the bounded case (i.e s^k are finite) we can set the marginal differential functions to $\Delta\gamma^j(n) = \infty$, where $n \geq s^j + 1$ and run BGA according to these new weights. In every iteration, BGA finds a shortest path that can either the edge (x, y) with a zero weight (and therefore by the non-negative stopping rule BGA terminates) or a shortest path that passes through areas node a^j, which must have negative weight.

Assume that in the first s iterations BGA finds only negative shortest paths through areas node a^j. Then according to Lemma A.2 every edge (x, a^j) in these shortest paths satisfied $f^j \leq s^j$. Otherwise, the edges (x, a^j) have infinity weight, and the shortest path does not passes them. Since $\sum_{j=1}^k f^j$ equals to the number of iterations (which is s), then for every area j we have $f^j = s^j$. Thus, in the next iteration of BGA the shortest path must be the edge (x, y), and BGA terminates.

Thus, after at most s iterations BGA terminates, and its complexity will be at most $O(smk)$ for the bounded case.

Let f flow be a flow defined on the graph $G =< V, E >$, and assume the reduced weight function w^{π} is defined on $G_f =< V, E_f >$, such that w^{π} **does not** satisfies the non-negative reduced weight property.

An edge $e \in E_f$ in the residual graph G_f is called *out-of-kilter* edge if its reduce weight is negative, i.e $w_f^{\pi}(e) < 0$. For those edges we define its kilter number $k(e)$ to be its residual capacity $k(e) = c_f(e)$. Note that if the kilter number must be different then zero; otherwise, the residual capacity is zero, and the edge weight $w_f(e)$ is set to infinite. For an in-kilter edge the kilter number is set to zero.

The out-of-kilter-algorithm (Algorithm 2) finds in every iteration an out-of-kilter edge and decreases its kilter number. In [18] they proved the following Lemma.

Lemma A.5: Let f flow be a flow defined on the graph $G =< V, E >$ and let π be an arbitrary node-potentials function. Suppose we run the out-of-kilter algorithm on these parameters, and let $e_i = (u, v)$ and π_i be respectively the out-of-kilter edge chosen in Step2 and node potentials of the i^{th} iteration (defined in step 5). Then the following claims take place:

1) If e is an out-of-kilter edge in the cycle w of Step 7 respect to node potentials π_i, then its kilter number is strictly decreases. Moreover, the kilter number of e_i strictly decreases.
2) All edges e in the cycle w have non-positive reduce weights (i.e. $w^{\pi}(e) \leq 0$). Moreover, e is an in-kilter edge in the cycle w, then its reduce weight $w^{\pi_i}(e)$ equals to zero.
3) The kilter number of every edge in the residual graph does not increased.

Of course, combining the previous lemma with the non-negative reduced weight condition proves the optimality of the algorithm.

Algorithm 2 The out-of-kilter-algorithm

```
Require: A feasible flow $f$ on $G$, a residual graph $G_f$, node potentials $\pi$, a source $x$ and a sink $y$ in $G$.
1: while the network contains an out-of-kilter edge in $G_f$ do
2:     Select an out-of-kilter edge $(u, v)$ in $G$.
3:     Define the length of each arc $e$ in $G_f$ as $\max 0, w^{\pi}(e)$.
4:     Let $d()$ denote the shortest path distances from node $v$ to all other nodes in $G_f - \{(u, v)\}$ and let $p$ denote
       a shortest path from node $v$ to node $u$
5:     Update $\pi(v) = \pi(v) - d(v)$ for every vertex $v$
6:     if $w^{\pi}(u, v) < 0$ then
7:         Define the cycle $w = p \bigcup (u, v)$
8:         Find $\delta = \min\{c(e) | e \in E\})$.
9:         Augment $\delta$ units of flow through $f$.
10:    end if
11: end while
```