Risk Aware Stochastic Placement of Cloud services: The Multiple Data Center Case[*]

Galia Shabtai[1], Danny Raz[2], and Yuval Shavitt[3]

1 School of Electrical Engineering, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel
galiashabtai@gmail.com
2 Faculty of Computer Science, The Technion, Haifa 32000, Israel
danny@cs.technion.ac.il
3 School of Electrical Engineering, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel
shavitt@eng.tau.ac.il

Abstract

Allocating the right amount of resources to each service in any of the data centers in a cloud environment is a very difficult task. In a previous work we considered the case where only two data centers are available and proposed a stochastic based placement algorithm to find a solution that minimizes the expected total cost of ownership. This approximation algorithm seems to work well for a very large family of overflow cost functions, which contains three functions that describe the most common practical situations. In this paper we generalize this work for arbitrary number of data centers and develop a generalized mechanism to assign services to data centers based on the available resources in each data center and the distribution of the demand for each service. We further show, using simulations based on real data that the scheme performs very well on realistic service workloads.

1998 ACM Subject Classification Stochastic programming, Nonlinear approximation

Keywords and phrases Approximation algorithms, Combinatorial optimization, Stochastic optimization, non-convex optimization, Cloud

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.

1 Introduction

The recent rapid development of cloud technology gives rise to "many-and-diverse" services being deployed in datacenters across the world. The placement of services to the available datacenters in the cloud has a critical impact on the ability to provide a ubiquitous cost-effective high quality service. There are many challenges associated with optimal service placement due to the large scale of the problem, the need to obtain state information, and the geographical spreading of the datacenters and users.

One intriguing problem is the fact that resource requirement of services changes over time and is not fully known at the time of placement. Moreover, while the average demand may follow a clear daily pattern, the actual demand of a service at a specific time may vary considerably according to the stochastic nature of the demand. One way of addressing this important problem is over-provisioning, that is, allocating resources according to the

[*] This paper was supported in part by the Neptune Consortium, Israel and by the Israeli Ministry of Science, Technology and Space.

peak demand. Clearly, this is not a cost effective approach and much of the resources are unused most of the time. A more economical approach, relying on the stochastic nature of the demand, is to allocate just the right amount of resources and potentially use additional mechanisms (such as diverting the service request to a remote location or dynamically buying additional resources) in case of overflow situations where demand exceeds the capacity. Clearly, the cost during such (overflow) events is higher than the normal cost. Moreover, in many cases it is proportional to the amount of unavailable resources. Obviously, the quantitative way of modeling the cost of an overflow situation, considerably depends on the actions taken (or not taken) in such cases. For example one may want to minimize the probability of an overflow event, or to minimize the expected overflow of the demand. The challenge is therefore to find an optimal placement for a given cost function.

The simplest, non-trivial variant is when there are two data centers in the cloud. This case was studied by us [6] where we combined techniques from two different lines of work, stochastic resource allocation ([3, 2, 7, 1]) and convex optimization ([5, 4]), to give an efficient robust algorithm for approximating the optimal solution. In this paper we extend this work to the general case of arbitrary number of data centers.

Our algorithm for the general case with multiple data centers follows the intuition explained in [6] for two data centers: the algorithm is trying to allocate more spare capacity to the high risk services. In our case this intuition translates to "The double-sorting algorithm" which sorts the data centers by their capacity, the services by their risk (defined as the amount of variance per unit of expectation) and then allocates consecutive segments of the sorted list of services to the data centers, with lower risk services allocated to smaller capacity data centers.

A challenge is finding the right partition of the sorted list to k consecutive segments, where k is the number of data centers (bins). In the two data centers case (where $k = 2$) this is done by a brute force search over the n possible partition points, and gives an almost linear time algorithm. When there are more than two data centers ($k > 2$) the brute force algorithm has complexity of about n^{k-1} and is impractical. The main bulk of the paper is devoted to describing an efficient algorithm for this problem and proving its correctness and complexity.

As explained before we are interested in giving a *generic* solution to the problem that is applicable to a wide class of cost functions. In the two-bin case [6] we showed how to solve the approximation problem for any cost function that respects three natural properties (see Section 4). In Section 6 we develop a dynamic programming algorithm that solves the k-bin case in time of about $O(n^3)$ for all cost functions in the above class.

Admittedly, while $O(n^3)$ gives a polynomial time algorithm independent of k, it may still be too expensive to run in practice, and therefore we next concentrate on presenting an efficient algorithms for specific, natural cost functions. We focus on three natural cost functions that are used in practice, which are SP-MED (stochastic placement with minimum expected deviation), SP-MWOP (stochastic placement with minimum worst overflow probability) and SP-MOP (stochastic placement with minimum overflow probability).

In Section 7 we devise a novel algorithm, the *moving sticks algorithm*, that uses a completely different approach for the problem. We formally analyze the algorithm for the SP-MWOP cost function, and show that it approximates the optimal solution in almost linear time. We also devise a similar algorithm for the other two cost functions, but we do not give a formal proof of correctness.

We implemented our algorithms and evaluated their performance over real data obtained from a mid-size operational data center and emulated workloads. Our results indicate that the new algorithms achieve a considerable gain when compared with commonly used naive solutions.

2 Problem formulation

The input to the problem consists of k and n, specifying the number of bins and services, and integers $\{c_j\}_{j=1}^k$, specifying the bin capacities. We are also given a partial description of n *independent* random variables $X = (X^{(1)}, \ldots, X^{(n)})$. This partial description includes the mean $\mu^{(i)}$, variance $V^{(i)} = \mathbb{E}(|X^{(i)} - \mu^{(i)}|^2)$ and $\rho^{(i)} = \mathbb{E}(|X^{(i)} - \mu^{(i)}|^3)$ of each variable $X^{(i)}$ ($\rho^{(i)}$ are needed only for the error estimation of the reduction to the normal case). The output is a partition of $[n]$ to k disjoint sets $S = S_1, \ldots, S_k \subseteq [n]$, where S_j includes indices of services that needs to be allocated to bin j. Our goal is to find a partition minimizing a given cost function $D(S)$.

In [6] we used the Berry-Esseen theorem (which is a quantitative version of the central limit theorem) to bound the gap between the optimal solution under independent *arbitrary* input demand distributions and the optimal solution under independent *normal* input demand distributions. We showed that when the number of services is large and the first three moments of the services satisfy a mild condition, there is a reduction from the general case, where the independent $X^{(i)}$ are almost arbitrary, to the normal case. The same reduction also applies in our case (when $k > 2$). Therefore, in this work we assume w.l.o.g. that the independent input services are normally distributed and in this case we do not need $\rho^{(i)}$.

3 Summary of our results for two data centers

In this section we present the main results of the normal two bin case, studied in [6], which are needed for the understanding of the $k > 2$ case analysis presented in the sequel.

The input to the problem is $c_1, c_2, \{\mu^{(i)}, V^{(i)}\}_{i=1}^n$ as before. Define $a^{(i)} = \frac{\mu^{(i)}}{\mu}$, $b^{(i)} = \frac{V^{(i)}}{V}$ and $P^{(i)} = (a^{(i)}, b^{(i)})$. If we split the services according to the partition $S = (S_1 = I, S_2 = [n] \setminus I)$, then the first bin is normally distributed with mean $\mu_1 = \mu \sum_{i \in I} a^{(i)}$ and variance $V_1 = V \sum_{i \in I} b^{(i)}$. Therefore, $(\frac{\mu_1}{\mu}, \frac{V_1}{V}) = P_I = \sum_{i \in I} P^{(i)}$. The point P_I also determines the mean and variance of the second bin: $(\frac{\mu_2}{\mu}, \frac{V_2}{V}) = (1,1) - P_I$. Hence, we can associate the partition S with the point P_I.

In [6] we defined a function $D : [0,1]^2 \to \mathbf{R}$ where $D(a,b)$ is the cost function D under a partition where the demand to the first bin is normally distributed with mean $a\mu$ and variance bV. We also decoupled the optimization problem into two separate and almost orthogonal problems: the first is understanding the feasible set of solutions, and the second is the behavior of the objective cost function as a *continuous* function over the two-dimensional domain.

A point P_I that is associated with a partition $S = (I, [n] \setminus I)$ is called *an integral point*. The convex hull of all the integral points is the set of all fractional points. The first task is to find a convenient description of this convex set.

▶ **Definition 1.** (The sorted paths) Sort the services by their variance to mean ratio (VMR) in increasing order (i.e., $\frac{V^{(1)}}{\mu^{(1)}} \leq \frac{V^{(2)}}{\mu^{(2)}} \leq \cdots \leq \frac{V^{(n)}}{\mu^{(n)}}$) and calculate the $P^{(1)}, P^{(2)}, \ldots, P^{(n)}$ vectors. For $i = 1, \ldots, n$ define

$$\begin{aligned} P_{bottom}^{[i]} &= P^{(1)} + P^{(2)} + \ldots + P^{(i)} \text{ and,} \\ P_{up}^{[i]} &= P^{(n)} + P^{(n-1)} + \ldots + P^{(n-i+1)}, \end{aligned}$$

and also define $P_{bottom}^{[0]} = P_{up}^{[0]} = (0,0)$.

The *bottom sorted path* is the curve that is formed by connecting $P_{bottom}^{[i]}$ and $P_{bottom}^{[i+1]}$ with a line, for $i = 0, \dots, n-1$. The *upper sorted path* is the curve that is formed by connecting $P_{up}^{[i]}$ and $P_{up}^{[i+1]}$ with a line, for $i = 0, \dots, n-1$.

The integral point $P_{bottom}^{[i]}$ on the bottom sorted path corresponds to allocating the i services with the lowest VMR to the first bin and the rest to the second. Similarly, the integral point $P_{up}^{[i]}$ on the upper sorted path corresponds to allocating the i services with the highest VMR to the first bin and the rest to the second. A fractional partition is one that allows splitting a service between several bins. [6] showed that:

▶ **Lemma 2.** *The polygon confined by the bottom sorted path and the upper sorted path is the convex hull of all possible partitions, i.e. all the fractional and integral points lie within it.*

The second task is analyzing the cost function D. We proved that when the cost function D respects three natural properties (see Section 4 below) and when $c_1 \le c_2$ the optimal fractional solution lies on the bottom sorted path and it splits at most one service between the two bins. The implication of that is that it is always better to allocate the low risk services to the smaller capacity bin and the high risk services to the higher capacity bin.

4 Three cost functions

Any cost function $D(a,b)$ that has the following three properties falls into the framework of [6]. We give here a brief description of the properties. Full details can be found in [6].

1. Symmetry: $D(a,b) = D(1 - a - \frac{c_2-c_1}{\mu}, 1-b)$.
2. Uni-modality in a: For every fixed $b \in [0,1]$, $D(a,b)$ has a unique minimum on $a \in [0,1]$, at some point $a = m(b)$. The curve $\{(m(b), b)\}$ is called the *valley*.
3. Central saddle point: D has a unique maximum over the valley at the point $(m(\frac{1}{2}), \frac{1}{2})$.

The above three properties hold for a very large family of cost functions, and in particular for the following three cost functions which are often used in practice. Again, we give here a brief description. Full details and proofs (that these cost functions indeed respect these three properties) can be found in [6].

- **SP-MED** (*Stochastic Placement with Min Expected Deviation*): In SP-MED the cost is the sum of the expected deviations of the bins, i.e. $\sum_{j=1}^{k} \mathbb{E} f_j(X_j)$, where $f_j(x)$ is the deviation function of bin j, i.e., $f_j(x) = x - c_j$ if $x > c_j$ and 0 otherwise, and $\mathbb{E} f_j(X_j)$ is the expected deviation of bin j.
- **SP-MWOP** (*Stochastic Placement with Min Worst Overflow Probability*): In SP-MWOP the cost is the minimal probability p, such that for every bin the probability that the bin overflows is at most p. Namely, if OF_j is the event that bin j overflows, then the cost of a placement is $\max_{j=1}^{k} \Pr[OF_j]$.
  In the normal case the overflow probability of bin j, denoted by OFP_j, is $OFP_j = 1 - \Phi(\Delta_j)$, where Φ is the cumulative distribution function of the standard normal distribution, $\Delta_j = \frac{c_j - \mu_j}{\sigma_j}$, $\mu_j = \sum_{i \in S_j} \mu^{(i)}$, and $\sigma_j = \sqrt{V_j} = \sqrt{\sum_{i \in S_j} V^{(i)}}$. Thus, $D_{SP-MWOP} = \max_{j=1}^{k} \{1 - \Phi(\Delta_j)\}$.
  We proved in [6] that when $k = 2$, the optimal fractional solution for the SP-MWOP cost function is the unique point on the bottom sorted path, in which $\Delta_1 = \Delta_2$ is satisfied.

- **SP-MOP** (*Stochastic Placement with Minimum Overflow Probability*): In SP-MOP the cost is the probability that *any* bin overflows, i.e. $\Pr[\bigcup_{j=1}^{k} OF_j]$. The total overflow probability is therefore $1-\prod_{j=1}^{k}(1-OFP_j)$, where in the normal case, $OFP_j = 1-\Phi(\Delta_j)$. Thus, $D_{SP-MOP} = 1 - \prod_{j=1}^{k} \Phi(\Delta_j)$.

5 The double sorting framework for more than two data centers

We now analyze the general $k > 2$ bin case using the results obtained for the two-bin case. We assume that the cost function has the following property:

The Local Optimality condition: In any optimal solution to a k-bin problem, the allocation for any two bins is also optimal. Formally, if $S_1, \dots, S_k$ is optimal for a k-bin problem, then for any j and j', the partition $S_j, S_{j'}$ is optimal for the two-bin problem defined by the services in $S_j \cup S_{j'}$ and capacities $c_j, c_{j'}$.

▶ **Theorem 3.** *The cost functions SP-MED, SP-MWOP and SP-MOP respect the local optimality condition.*

Proof. Let $S_1, \dots, S_k$ be an optimal solution. Suppose there are j and j' for which $S_j, S_{j'}$ are not the optimal solution for the two-bin problem defined by the services in $S_j \cup S_{j'}$ and capacities $c_j, c_{j'}$. Change the allocation of the solution $S_1, \dots, S_k$ on bins j and j' to an optimal solution for the two bin problem.

- In SP-MWOP, this change improves the worst overflow probability of the two bins while not affecting the overflow probability of any other bin.
- In SP-MED, this change improves the expected total deviation of the two bins while not affecting the expected deviation of any other bin.
- In SP-MOP, this change improves $(1-OFP_j)(1-OFP_{j'})$ while not affecting the overflow probability of any other bin.

In total we get a better solution, in contradiction to the optimality of $S_1, \dots, S_k$. ◀

▶ **Definition 4.** (A sorted solution) Assume the bins are sorted by their capacity, $c_1 \le c_2 \le \dots \le c_k$. A fractional solution is called *sorted* if for every $j < j'$ and every two services i and i' such that service i (resp. i') is allocated to bin j (resp. j') it holds that the VMR of service i is not larger than that of service i'.

▶ **Theorem 5.** *If the cost function satisfies the symmetry, unimodality in a, central saddle point and local optimality properties, then the optimal fractional solution is* sorted.

Proof. Assume there is an optimal solution $S_1, \dots, S_k$ that is *not* sorted. I.e., there exist $j < j'$ and i, i' such that service i (resp. i') is allocated to bin j (resp. j') and the VMR of service i is strictly larger than that of service i'. By [6] the sorted fractional solution for the problem defined by bins j and j' is *strictly better* than the one offered by the solution $S_1, \dots, S_k$. This means the solution is not optimal for bins j and j' - in contradiction to the local optimality condition. ◀

Next we present an algorithmic framework for the problem:

The double sorting algorithm framework

- Sort the bins by their capacity $c_1 \le c_2 \le \ldots \le c_k$.
- Sort the services by their VMR $\frac{V^{(1)}}{\mu^{(1)}} \le \frac{V^{(2)}}{\mu^{(2)}} \le \cdots \le \frac{V^{(n)}}{\mu^{(n)}}$.
- Use a partitioning algorithm to find $k-1$ partition points $\ell_0 \triangleq 0 \le \ell_1 < \ldots \ell_{k-1} \le \ell_k \triangleq n$ and output $S_1, \ldots, S_k$ accordingly.
- Allocate the services in S_j to bin number j (with capacity c_j).

The algorithmic framework is not complete since it does not specify how to find the partition points on the bottom sorted path. With two bins there is only one partition point, and we can check all possible $n-1$ integral partition points. With k bins there are $\binom{n}{k-1}$ possible partition points, and checking all partition points may be infeasible. In the sequel, we present three algorithms for finding a good partition and show (using the local optimality condition) that if we know how to solve the two bin case, we can also solve the k-bin case.

6 A dynamic programming algorithm

We now describe a dynamic programming algorithm that finds the best integral solution on the bottom sorted path. It assumes we can solve the $k=2$ case (for any cost function D). Thus, the algorithm is quite general and may be seen as a general reduction from arbitrary k to the $k=2$ case. The complexity of the algorithm is polynomial, on the order of about n^3 time.

Finding an integral partition using dynamic programming

For each $1 \le t \le \log k$, $1 \le i < i' \le n$ and $j = 1, \ldots, k$ keep the two values $Partition(2^t, i, i', j)$ and $D(2^t, i, i', j)$ that are defined as follows:

- Base case $t=1$: $Partition(2, i, i', j)$ is the best integral partition point for the two bin problem with inputs $X_i, \ldots, X_{i'}$ and capacities c_j, c_{j+1}. $D(2, i, i', j)$ is its corresponding cost.
- Induction step. Suppose we have built the tables for t, we show how to build the tables for $t+1$. We let $D(2^{t+1}, i, i', j)$ be: $\min\{F(D(2^t, i, i''-1, j), D(2^t, i'', i', j+2^t))\}$, where the minimization is over $i + 2^t \le i'' \le i' - (2^t - 1)$ and F depends on the cost function we are dealing with[a]. We let $Partition(2^{t+1}, i, i', j)$ be the partition point that obtains the minimum in the above equation.

The best partition to k bins (assuming k is a power of two) can be recovered from the tables. For example for 4 bins $Partition(4, 1, n, 1)$ returns the middle point ℓ_2 of the partition, and the partition points within each half are obtained by $\ell_1 = Partition(2, 1, \ell_2 - 1, 1)$ and $\ell_3 = Partition(2, \ell_2, n, 3)$.

[a] for SP-MED: $F(x,y) = x+y$; for SP-MWOP: $F(x,y) = max\{x,y\}$; for SP-MOP: $F(x,y) = 1-(1-x)\cdot(1-y)$.

It can be seen (by a simple induction) that $Partition(2^t, i, i', j)$ gives the middle point of the best integral solution on the sorted path to the problem of dividing $X_i, \ldots, X_{i'}$ to 2^t bins with capacities $c_j, c_{j+1}, \ldots, c_{j+2^t-1}$. It can also be verified that the running time of the algorithm is $O(n^3 k \log k)$.[1]

7 The moving sticks (MVS) algorithm for SP-MWOP

The main disadvantage of the dynamic algorithm is that it runs in about n^3 time. For specific cases, such as SP-MWOP cost function, the optimal solution may be found much faster.

We proved in [6] that when $k = 2$, the optimal fractional solution for SP-MWOP cost function is the unique point on the bottom sorted path, in which the equality $\Delta_1 = \Delta_2$ is satisfied. Hence, by the local optimality condition we can say that the optimal fractional solution of SP-MWOP for k-bins is obtained on a partition where $\Delta_1 = \Delta_2 = \ldots = \Delta_k$.

The dynamic algorithm finds an integral solution. In the scope of the moving sticks (MVS) algorithm, it is more convenient to find a fractional solution and work with fractional partition point values. Suppose we have n services sorted by their VMR, and service i has mean $\mu^{(i)}$. The first m services correspond to the fractional point $f = \frac{\sum_{i=1}^{m} \mu^{(i)}}{\mu} \in [0, 1]$. Conversely, a point $f \in [0, 1]$ with $\frac{\sum_{i=1}^{m} \mu^{(i)}}{\mu} \leq f < \frac{\sum_{i=1}^{m+1} \mu^{(i)}}{\mu}$ corresponds to taking the first m services, and the appropriate part from service $m+1$. Similarly, a segment $[\alpha_1, \alpha_2] \subseteq [0, 1]$ corresponds to a consecutive set of services, with up to two incomplete services (from the beginning and end of the segment). A fractional partition is $0 = f_0 < f_1 < \ldots < f_{k-1} < f_k = 1$.

Following is an informal high level description of the algorithm: At each time step of the algorithm we have $k - 1$ sticks positioned between the n services (sorted in increasing order by their VMR) and thus separating them to k bins, where the j'th segment is allocated to the j'th bin. At the beginning, we fill the last bins to full capacity as much as we can. After initialization, there are some completely full bins at the end of the bin list, at most one bin which is partially full, and zero or more empty bins at the beginning of the list. It is easy to see that in the initial state each partition point is left to its position in the fractional optimal solution. Also, at the first step $\Delta_1 \geq \Delta_2 \geq \ldots \geq \Delta_k$. Then, at each time step, we make a local improvement by fractionally moving one of the sticks right, provided that this movement does not break the invariance that $\Delta_1 \geq \Delta_2 \geq \ldots \geq \Delta_k$. We will see that this guarantees that again each partition point is left to its position in the fractional optimal solution. The process then continues until no local improvement is possible. Using the fact that the SP-MWOP fractional optimal partition for k bins is achieved when $\Delta_1 = \Delta_2 = \ldots = \Delta_k$, we prove that the process converges to the optimal fractional solution on the bottom sorted path.

The fractional step size τ, is the fraction of the total mean, μ, moved from one bin to its predecessor in one step of the algorithm. τ determines the algorithm running time as well as its accuracy. Since there are $k - 1$ sticks, and each stick can move at most $\frac{1}{\tau}$ times, the algorithm has at most $\frac{k-1}{\tau}$ steps before it stops. Moreover, the smaller τ is the better the accuracy is (and, correspondingly, the running time gets larger). In Theorem 6 we show that the algorithm error bound linearly depends on k and τ.

[1] The best integral solution on the bottom sorted path is not necessarily the optimal integral solution. However, in any reasonable situation, its cost is *close* to the optimal cost, and even to the optimal fractional solution. To see that notice that by [6] the optimal fractional solution is on the bottom sorted path, and if there is no dominant service (see [6] for a rigorous definition) there must be an integral point on the bottom sorted path that is close to the optimal fractional solution, and therefore by continuity (if indeed the cost function is continuous) the cost of the best integral solution on the bottom sorted path is close to the optimal fractional cost. A rigorous analysis for two bins and the cost functions SP-MED and SP-MWOP is presented in [6] . The k bin case is a straight forward extension of these results.

The moving sticks algorithm for SP-MWOP: Initialization

```
- Initialization (at time $t = 0$): /* completely fill up bins */
    - Let $\widetilde{j}$ be the smallest bin index which satisfies $c_{\widetilde{j}+1} + c_{\widetilde{j}+2} + \ldots + c_k < \mu$.
    - $\forall j = 0, 1, \ldots, k$, let
          $f_j^{(0)} = \begin{cases} 0 & \text{If } j < \widetilde{j} \\ 1 - \frac{c_{j+1}+c_{j+2}+\ldots+c_k}{\mu} & \text{Else} \end{cases}$
    - Let $I^{(0)} = \{2, \ldots, k\}$.[a]
```

[a] We could have taken $I^{(0)} = \left\{\widetilde{j}, \widetilde{j}+1\right\}$, but this does not improve the asymptotic behavior of the algorithm, and we choose the simpler rule.

A detailed description of the moving sticks algorithm for the SP-MWOP is given below. We later prove its correctness. For the description we let $\Delta_j^{[\alpha_1,\alpha_2]}$ denote the value of Δ_j, when bin j is (fractionally) allocated the items corresponding to the segment $[\alpha_1, \alpha_2] \subseteq [0, 1]$. We also let $I^{(t)}$ denote the set of bin indices that can potentially move $\tau \cdot \mu$ to their predecessor without violating the invariance.

The moving sticks algorithm for SP-MWOP: Main loop

```
- Main loop: Start with $t = 0$. Repeat until $I = \emptyset$:
    - Pick an index $j \in I^{(t)}$.
    - Calculate:
        ◇ $next\Delta_{j-1} = \Delta_{j-1}^{[f_{j-2}^{(t)}, f_{j-1}^{(t)}+\tau]}$,
        ◇ $next\Delta_j = \Delta_j^{[f_{j-1}^{(t)}+\tau, f_j^{(t)}]}$.
    - If ($next\Delta_j > next\Delta_{j-1}$) /* we break the invariance */
        ◇ Remove index $j$ from $I$. /* do not move any stick */
    - Else,
        ◇ Set $f_{j-1}^{(t+1)} = f_{j-1}^{(t)} + \tau$ and keep all other values unchanged, i.e., $f_{j'}^{(t+1)} = f_{j'}^{(t)}$, $\forall$
          $j' \neq j - 1$ .
        ◇ Let
          $I^{(t+1)} = \begin{cases} I^{(t)} \cup \{3\} & \text{If } j = 2 \\ I^{(t)} \cup \{k-1\} & \text{If } j = k \\ I^{(t)} \cup \{j-1, j+1\} & \text{Else} \end{cases}$
    - Increase $t$.
- Stopping rule: When $I = \emptyset$ stop and output $f_1^{(t)}, \ldots, f_{k-1}^{(t)}$.
```

We now prove the correctness of the moving sticks algorithm for the SP-MWOP problem.

▶ **Theorem 6.** *Let $0 = f_0^* \leq f_1^* \leq f_2^* \leq \ldots \leq f_{k-1}^* \leq f_k^* = 1$ be the optimal fractional partition. The following is true for the moving sticks algorithm, when considering the SP-MWOP problem:*

1. *At time $t = 0$, no partition point is positioned right to its optimal position, i.e., $f_j^{(0)} \leq f_j^*$, $\forall j \in [0, k]$*
2. *At any time step $t \geq 0$, each separator may move only rightward, i.e., $f_j^{(t+1)} \geq f_j^{(t)}$, $\forall j \in [1, k-1]$*
3. *For $t \geq 0$, define $\Delta_j^{(t)} = \Delta_j^{[f_{j-1}^{(t)}, f_j^{(t)}]}$. At any time step $t \geq 0$,*

$$\Delta_1^{(t)} \geq \Delta_2^{(t)} \geq \ldots \geq \Delta_k^{(t)}. \tag{1}$$

4. *If $f_j^{(t)} \leq f_j^* \ \forall j \in [1, k-1]$, then also $f_j^{(t+1)} \leq f_j^* \ \forall j \in [1, k-1]$.*
5. *Stopping state: when we stop, we cannot move any stick right by τ without breaking the algorithm invariance (1).*
6. *Error bounding: Let VMR_{max} be the maximum VMR value of the input, i.e. $VMR_{max} = \frac{V^{(n)}}{\mu^{(n)}}$, and let b_{min} be the minimum variance portion a bin gets. The difference between the cost of the fractional solution found by the moving sticks algorithm and the optimal fractional solution is at most $(k-1)\epsilon$, where $\epsilon = \frac{c}{\sigma} \cdot \frac{\sqrt{2}(1+VMR_{max})}{b_{min}\sqrt{b_{min}}} \cdot \tau$.*

For the proof of Theorem 6 we need two technical lemmas.

▶ **Lemma 7.** *(monotonicity of an optimal partition point) Let $OPT_j^{[\alpha_1,\alpha_2]}$ denote the SP-MWOP fractional partition point which optimally allocates the items that belong to the segment $[\alpha_1, \alpha_2]$ to the two bins j and $j+1$. Let $1 \leq j \leq k-1$. Then:*
1. *For all $\alpha_1 \leq \alpha_2 < \alpha_3 \leq 1$, $OPT_j^{[\alpha_1,\alpha_3]} \leq OPT_j^{[\alpha_2,\alpha_3]}$.*
2. *For all $\alpha_1 < \alpha_2 \leq \alpha_3 \leq 1$, $OPT_j^{[\alpha_1,\alpha_2]} \leq OPT_j^{[\alpha_1,\alpha_3]}$.*

Proof. We prove the first item, the proof of the second is similar. Recall that an SP-MWOP fractional optimal partition for two bins is achieved at the point in which $\Delta_1 = \Delta_2$, i.e., $\Delta_j^{[\alpha_2, OPT_j^{[\alpha_2,\alpha_3]}]} = \Delta_{j+1}^{[OPT_j^{[\alpha_2,\alpha_3]}, \alpha_3]}$. If we add the items in the segment $[\alpha_1, \alpha_2]$ to bin j, we decrease its spare capacity and get that $\Delta_j^{[\alpha_1, OPT_j^{[\alpha_2,\alpha_3]}]} < \Delta_j^{[\alpha_2, OPT_j^{[\alpha_2,\alpha_3]}]}$ and therefore, $\Delta_j^{[\alpha_1, OPT_j^{[\alpha_2,\alpha_3]}]} < \Delta_{j+1}^{[OPT_j^{[\alpha_2,\alpha_3]}, \alpha_3]}$. In order to reach the optimal partition in the segment $[\alpha_1, \alpha_3]$, we need to decrease Δ_{j+1} and increase Δ_j till we get equality, i.e., we need to move items from bin j to bin $j+1$, by decreasing the partition point between them. Hence, we get that $OPT_j^{[\alpha_1,\alpha_3]} \leq OPT_j^{[\alpha_2,\alpha_3]}$. ◀

▶ **Lemma 8.** *(continuity of Δ) Let VMR_{max}, b_{min} as before. Let $P_1 = (\alpha_1, \beta_1)$, $P_1' = (\alpha_1', \beta_1')$, $P_2 = (\alpha_2, \beta_2)$ and $P_2' = (\alpha_2', \beta_2')$, be four points on the bottom sorted path, such that $0 \leq \alpha_1 < \alpha_1' < \alpha_2 < \alpha_2' \leq 1$ (and therefore, $0 \leq \beta_1 \leq \beta_1' \leq \beta_2 \leq \beta_2' \leq 1$), and also $\alpha_1' - \alpha_1 = \tau$ and $\alpha_2' - \alpha_2 = \tau$. Then, for all $1 \leq j \leq k$: $\Delta_j^{[\alpha_1,\alpha_2]} - \Delta_j^{[\alpha_1,\alpha_2']} \leq \epsilon$, and, $\Delta_j^{[\alpha_1,\alpha_2']} - \Delta_j^{[\alpha_1',\alpha_2']} \leq \epsilon$, where $\epsilon = \frac{c}{\sigma} \cdot \frac{\sqrt{2}(1+VMR_{max})}{b_{min}\sqrt{b_{min}}} \cdot \tau$*

Proof. We recall a notation we have used in Section 3: $\Delta_j(a,b) = \frac{c_j - a\mu}{\sigma\sqrt{b}}$, where the demand to bin j is normally distributed with mean $a\mu$ and variance bV. In this notation, and using the mean value theorem:

$$\begin{aligned} \Delta_j^{[\alpha_1,\alpha_2]} - \Delta_j^{[\alpha_1,\alpha_2']} &= \Delta_j(\alpha_2 - \alpha_1, \beta_2 - \beta_1) - \Delta_j(\alpha_2' - \alpha_1, \beta_2' - \beta_1) \leq \qquad (2) \\ & |\nabla(\Delta_j(a,b))| \cdot |(\alpha_2 - \alpha_1, \beta_2 - \beta_1) - (\alpha_2' - \alpha_1, \beta_2' - \beta_1)|, \qquad (3) \end{aligned}$$

where $\alpha_2 - \alpha_1 \leq a \leq \alpha_2' - \alpha_1$ and $\beta_2 - \beta_1 \leq b \leq \beta_2' - \beta_1$.

It is easy to bound the gradient of $\Delta(a,b)$:
$|\frac{\partial \Delta_j(a,b)}{\partial a}| = \frac{\mu}{\sigma} \cdot \frac{1}{\sqrt{b}} \leq \frac{c}{\sigma} \cdot \frac{1}{\sqrt{b}}$
$|\frac{\partial \Delta_j(a,b)}{\partial b}| = \frac{1}{2} \cdot \frac{c_j - a\mu}{\sigma} \cdot \frac{1}{b\sqrt{b}} \leq \frac{c}{\sigma} \cdot \frac{1}{b\sqrt{b}}$
$|\nabla(\Delta_j(a,b))| \leq \frac{c}{\sigma} \cdot \frac{1}{\sqrt{b}} \cdot \sqrt{1 + \frac{1}{b^2}} = \frac{c}{\sigma} \cdot \frac{\sqrt{1+b^2}}{b\sqrt{b}} \leq \frac{c}{\sigma} \cdot \frac{\sqrt{2}}{b\sqrt{b}}$.
Since $b \geq b_{min}$ we get that: $|\nabla(\Delta_j(a,b))| \leq \frac{c}{\sigma} \cdot \frac{\sqrt{2}}{b_{min}\sqrt{b_{min}}}$.

Finally, $|(\alpha_2 - \alpha_1, \beta_2 - \beta_1) - (\alpha_2' - \alpha_1, \beta_2' - \beta_1)| \leq |\alpha_2' - \alpha_2| + |\beta_2' - \beta_2|$. We know that $|\alpha_2' - \alpha_2| \leq \tau$. Also, $\frac{\beta_2' - \beta_2}{\alpha_2' - \alpha_2} \leq VMR_{max}$. Therefore, we can say that $|(a', b') - (a'', b'')| \leq \tau + \tau \cdot VMR_{max} = \tau(1 + VMR_{max})$. When we plug this in Eq (3) we get the lemma. ◀

We are now ready to prove Theorem 6:

Proof. We go over the items one by one:

item (1): For every $0 < j \leq \widetilde{j}$ the algorithm sets $f_{j-1}^{(0)} = 0$, which is obviously less than or equal to the optimal value f_{j-1}^*. Also, for every $\widetilde{j} < j \leq k$, the algorithm initializes the j'th bin to be completely full. In all partitions (including the optimal fractional partition) no bin is allocated more than its capacity. Hence , again, $f_{j-1}^{(0)} \leq f_{j-1}^*$. Finally, $f_k^{(0)} = f_k^* = 1$.

item (2): At each time step, the algorithm increases one separator value by τ (i.e., moves this separator right) and leaves all others unchanged. Therefore, $f_j^{(t+1)} \geq f_j^{(t)}$, $\forall j \in [1, k-1]$

item (3): After initialization, $\Delta_j^{(0)} = 0$ for every completely full bin $\widetilde{j} < j \leq k$, $0 < \Delta_j^{(0)} < \infty$ for the only partially full bin $j = \widetilde{j}$, and $\Delta_j^{(0)} = \infty$ for every empty bin $j < \widetilde{j}$. Therefore, after initialization the inequality $\Delta_1^{(0)} \geq \Delta_2^{(0)} \geq \ldots \geq \Delta_k^{(0)}$ holds.

Let us assume that at the end of time $t > 0$ the inequality $\Delta_1^{(t)} \geq \Delta_2^{(t)} \geq \ldots \geq \Delta_k^{(t)}$ holds. At time $t+1$, the algorithm picks bin j' and checks whether to move stick $j'-1$ and update $f_{j'-1}$. Since all other sticks are not updated, $\Delta_j^{(t+1)} = \Delta_j^{(t)}$, $\forall j \notin \{j'-1, j'\}$. In addition, the stick is moved only if the inequality $\Delta_{j'-1}^{(t+1)} \geq \Delta_{j'}^{(t+1)}$ stays true after the move. And since the move decreases $\Delta_{j'}$ (i.e. $\Delta_{j'}^{(t+1)} \leq \Delta_{j'}^{(t)}$) and increases $\Delta_{j'-1}$ (i.e. $\Delta_{j'-1}^{(t+1)} \geq \Delta_{j'-1}^{(t)}$), the inequality $\Delta_1^{(t+1)} \geq \Delta_2^{(t+1)} \geq \ldots \geq \Delta_k^{(t+1)}$ is also true.

item (4): At each time step t, the algorithm picks bin $j' \in \{2, \ldots, k\}$ and moves a fraction of an item from bin j' to bin $j'-1$, as long as the invariance $\Delta_{j'-1} \geq \Delta_{j'}$ stays true after the move. All other bins are kept untouched and therefore $f_j^{(t+1)} = f_j^{(t)} \leq f_j^*$ $\forall j \neq j'-1$. Now, let us consider bin $j'-1$ and bin j'. The list of items packed in both bins at time t is the list of items which corresponds to the segment $[f_{j'-2}^{(t)}, f_{j'}^{(t)}]$ and the algorithm, by moving fraction of items from bin j' to bin $j'-1$, increases $f_{j'-1}^{(t)}$ provided that the inequality $\Delta_{j'-1} \geq \Delta_{j'}$ stays valid. Since the optimal fractional solution is achieved when $\Delta_{j'-1} = \Delta_{j'}$, we get that

$$f_{j'-1}^{(t+1)} \quad \leq \quad OPT_{j'-1}^{[f_{j'-2}^{(t)}, f_{j'}^{(t)}]}. \tag{4}$$

In the optimal fractional partition the list of items packed in bins $j'-1$ and j' correspond to the segment $[f_{j'-2}^*, f_{j'}^*]$. We are given that $f_j^{(t)} \leq f_j^*$ $\forall t$ and $j \in \{1, \ldots, k-1\}$, and in particular $f_{j'-2}^{(t)} \leq f_{j'-2}^*$ and $f_{j'}^{(t)} \leq f_{j'}^*$. By applying Lemma 7 twice (once with $\alpha = f_{j'-2}^{(t)}$, $\beta_1 = f_{j'}^{(t)}$ and $\beta_2 = f_{j'}^*$, and once with $\alpha_1 = f_{j'-2}^{(t)}$, $\alpha_2 = f_{j'-2}^*$ and $\beta = f_{j'}^*$) we conclude that

$$OPT_{j'-1}^{[f_{j'-2}^{(t)}, f_{j'}^{(t)}]} \quad \leq \quad f_{j'-1}^*. \tag{5}$$

Equations (4) and (5) together imply that $f_{j'-1}^{(t+1)} \leq f_{j'-1}^*$.

item (5): We will show that at any time t, bin indices which are not included in $I^{(t)}$ cannot move $\tau \cdot \mu$ to their preceding bin without breaking the algorithm invariance. At the beginning, $I^{(t=0)} = \{2, \ldots, k\}$, so there is no bin outside $I^{(t=0)}$. Assume the property is true for time t, and let us prove correctness for time $t+1$. At time t, the algorithm picks an index $j \in I$, and a fractional movement of items from bin j to bin $j-1$ is examined. If the movement breaks the invariance, no movement is done and j is justifiably removed from I, i.e. $I^{(t+1)} = I^{(t)} \setminus \{j\}$. If the movement does not break the invariance, then the move is executed ($f_{j'-1}^{(t+1)}$ is updated) and the bins that are affected (i.e., bin indices $j-1$ and $j+1$) are added to I. Hence the required property is preserved.

Hence, when we stop, $I^{(t)} = \emptyset$, i.e., a move of $\tau \cdot \mu$ from any bin j to bin $j-1$ will necessarily break the invariance, as we had to prove.

item (6): Let $0 = f_0^* \leq f_1^* \leq \ldots \leq f_k^* = 1$ be the partition points of the fractional optimal solution, and $0 = f_0 \leq f_1 \leq \ldots \leq f_k = 1$ be the partition points found by the moving sticks algorithm. Similarly, Δ_j^* (resp. Δ_j) is the j'th bin Δ value in the partition f^* (resp. f). In the partition f^* we have $\Delta_1^* = \ldots = \Delta_k^*$. We now show that when the moving sticks algorithm stops all the Δ_j values are close to each other. Namely,

▶ Claim 9. For every $1 \leq j \leq k-1$: $\Delta_j \leq \Delta_{j+1} + 2\epsilon$.

Proof. Let $f_j^{'}$ be the optimum partition point for the two bins problem j and $j+1$, with the item list in segment $[f_{j-1}, f_{j+1}]$. From the algorithm invariance we know that $\Delta_j^{[f_{j-1},f_j]} \geq \Delta_{j+1}^{[f_j,f_{j+1}]}$. On the other hand, $\Delta_j^{[f_{j-1},f_j^{'}]} = \Delta_{j+1}^{[f_j^{'},f_{j+1}]}$, since $f_j^{'}$ is an optimum partition point. Therefore, we get that $f_j \leq f_j^{'}$. Moreover, by item (5) $f_j^{'} - f_j < \tau$, and by Lemma 8, $\Delta_j^{[f_{j-1},f_j]} - \Delta_j^{[f_{j-1},f_j^{'}]} \leq \epsilon$ and $\Delta_{j+1}^{[f_j^{'},f_{j+1}]} - \Delta_{j+1}^{[f_j,f_{j+1}]} \leq \epsilon$. Therefore,

$$\Delta_j = \Delta_j^{[f_{j-1},f_j]} \leq \Delta_j^{[f_{j-1},f_j^{'}]} + \epsilon = \Delta_{j+1}^{[f_j^{'},f_{j+1}]} + \epsilon \leq \Delta_{j+1}^{[f_j,f_{j+1}]} + 2\epsilon = \Delta_{j+1} + 2\epsilon.$$

◀

The claim implies that $\Delta_k \geq \Delta_{k-1} - 2\epsilon \geq \Delta_{k-2} - 4\epsilon \geq \ldots \geq \Delta_1 - 2(k-1)\epsilon$. However, from theorem 6 we know that $f_{k-1} \leq f_{k-1}^*$ and we also know that $f_k = f_k^*$. This means that $[f_{k-1}^*, f_k^*] \subseteq [f_{k-1}, f_k]$, and hence $\Delta_k \leq \Delta_k^*$. Using similar arguments, we can say that $\Delta_1 \geq \Delta_1^*$. Together, $\Delta_k^* \geq \Delta_k \geq \Delta_1 - 2(k-1)\epsilon \geq \Delta_1^* - 2(k-1)\epsilon = \Delta_k^* - 2(k-1)\epsilon$. The algorithm invariance is $\Delta_1 \geq \Delta_2 \geq \ldots \geq \Delta_k$, and this is also true when the algorithm ends. The cost of the fractional solution found by the moving sticks algorithm is therefore $max_{j=1}^k\{1 - \Phi[\Delta_j]\} = 1 - \Phi[\Delta_k]$. Also, the cost of the fractional optimal solution is $1 - \Phi[\Delta_k^*]$. The error is therefore $\Phi[\Delta_k^*] - \Phi[\Delta_k]$, where we know that $|\Delta_k - \Delta_k^*| \leq 2(k-1)\epsilon$. By the mean value theorem the difference is at most $\phi[\Delta_\xi] \cdot 2(k-1)\epsilon \leq (k-1)\epsilon$, where $\Delta_k^* - 2(k-1)\epsilon \leq \Delta_\xi \leq \Delta_k^*$.

◀

8 The generalized moving sticks (GMVS) algorithm

In the previous section we introduced the moving sticks algorithm for the SP-MWOP cost function and gave a correctness proof as well as error analysis. In this section we present a more general form of the moving sticks algorithm for other cost functions, such as SP-MED and SP-MOP, without a correctness proof and without error bounding. Simulations show that the general moving stick (GMVS) algorithm performs extremely well for all three cost functions we consider in this paper.

A key observation for generalizing the moving sticks algorithm for other cost functions is that the cost on the bottom sorted path, when expressed as a function of a[2], is unimodal in the two bin case when $\Delta_1 \geq 0$ and $\Delta_2 \geq 0$, i.e., decreasing till the optimal point and increasing from there on. Therefore, we can leave the initialization as is, and change only part of the main loop, such that at each time step, there is a fractional move of one of the sticks right, provided that this movement decreases the two-bin cost function of the two bins

[2] where a is the portion of the total mean allocated to the first bin, i.e., $a = \frac{\mu_1}{\mu}$

left and right of it. The advantage of this generalized version is that it is applicable to many cost functions, such as SP-MED, SP-MWOP and SP-MOP.

Following is a detailed description of the generalized main loop. For the description, we let $D_j^{[\alpha_1,\alpha_2,\alpha_3]}$ denote the cost of the two bins j and $j+1$, where bin j is (fractionally) allocated the items corresponding to the segment $[\alpha_1,\alpha_2] \subseteq [0,1]$ and bin $j+1$ is (fractionally) allocated the items corresponding to the segment $[\alpha_2,\alpha_3] \subseteq [0,1]$.

The generalized moving sticks algorithm: Main loop

```
- Main loop: Start with $t = 0$. Repeat until $I = \emptyset$:
    - Pick an index $j \in I^{(t)}$.
    - Calculate:
        ◇ $currentD = D_{j-1}^{[f_{j-2}^{(t)}, f_{j-1}^{(t)}, f_j^{(t)}]}$.
        ◇ $nextD = D_{j-1}^{[f_{j-2}^{(t)}, f_{j-1}^{(t)}+\tau, f_j^{(t)}]}$,
    - If $(nextD > currentD)$ /* two-bin cost is increased */
        ◇ Remove index $j$ from $I$. /* do not move any stick */
    - Else,
        ◇ Set $f_{j-1}^{(t+1)} = f_{j-1}^{(t)} + \tau$ and keep all other values unchanged, i.e., $f_{j'}^{(t+1)} = f_{j'}^{(t)}$, $\forall$ $j' \neq j-1$ .
        ◇ Let
          $$I^{(t+1)} = \begin{cases} I^{(t)} \cup \{3\} & \text{If } j = 2 \\ I^{(t)} \cup \{k-1\} & \text{If } j = k \\ I^{(t)} \cup \{j-1, j+1\} & \text{Else} \end{cases}$$
    - Increase $t$.
- Stopping rule: When $I = \emptyset$ stop and output $f_1^{(t)}, \ldots, f_{k-1}^{(t)}$.
```

9 Conclusion

We present an analytical scheme for stochastic placement algorithms, using the stochastic behavior of the demand. We generalize the results of previous work, which considered only two data centers, and develop efficient, almost optimal algorithms that work for a family of target cost functions and any number of data centers. In particular, we solve SP-MED (that minimizes the expected deviation), SP-MOP (that minimizes the probability of overflow) and SP-MWOP (that guarantees that for every bin the probability it overflows is small). We believe the framework is applicable for many other natural cost functions. As in the case of only two data centers, the results in this paper hold for any large enough collection of independent services of whatever distribution.

We show that if we know how to solve the two bin case, we can also solve the $k > 2$ bin case and present three algorithms for finding a good placement. The first algorithm is a dynamic programming algorithm, which finds the best integral point on the bottom sorted path and runs in about n^3 time. The second, is the moving sticks algorithm for the SP-MWOP cost function. We prove that it converges to the optimal solution in almost linear time. The last algorithm is a general version of the moving sticks algorithm, which is applicable for other cost functions and also runs in almost linear time. However, we do not prove its correctness.

Our simulation results indicate that our algorithms provide considerable gain using real data from a mid-size operational data center, compared with commonly used naive solutions.

A Simulation Results

In this section we present our simulation results for k bins. In all simulations we used 4 bins with bin capacity ratio of $c_{i+1} = 2c_i$. For each cost function, we compared the generalized moving sticks (GMVS) algorithm with two algorithms we call BS (Balanced Spares) and BL (Balanced Load), as in [6]. The BS algorithm goes through the list, item by item, and allocates each item to the bin which has more available space. In this way, the spare capacity is balanced. On the other hand, the BL algorithm goes through the list, item by item, and allocates each item to the bin which is less loaded, i.e., the bin with higher $\frac{\text{available space}}{\text{bin capacity}}$ value . In this way, the bin load is balanced. The BL and BS algorithms are natural benchmarks and also much better than other naive solutions like first-fit and first-fit decreasing.

We show two different results for the GMVS algorithm: *Fractional GMVS* and *Integral GMVS*. The fractional GMVS algorithm uses the fractional partition output of the generalized moving sticks algorithm, as described in section 8. The integral GMVS algorithm takes this fractional partition and converts it to an integral partition by moving each stick left to next integral point.[3]

In the case of SP-MWOP cost function, we also added results for the *Fractional MVS* and *Integral MVS* algorithms. As before, the fractional MVS algorithm uses the fractional partition output of the moving sticks algorithm, as described in section 7, and the integral MVS algorithm takes this fractional partition and converts it to an integral partition by moving each stick left to next integral point.

We implemented one straight forward improvement for the runtime of both MVS and GMVS algorithms: before examining a fractional movement of an item, we examine the movement of the whole item at once. In this way, if the entire item can be moved from one bin to the other at a given stage, we move it in one step, instead of fractionally moving it in several consecutive steps. This improvement does not change the results of the algorithms, but it improves the running time. Note that we didn't check the extent of this improvement, since we only implemented this version of the algorithms.

A.1 Results for Synthetic Normally Distributed Data

The synthetic data for this part was generated exactly as explained in [6] for the 2 bin case, except that we generated only 400 elements instead of 500 elements because executing the dynamic algorithm (whose complexity is $O(n^3)$) takes too long on $n = 500$.

Figure 1 shows that for SP-MED and SP-MOP cost functions the Integral GMVS is very close to the Fractional GMVS, as expected. However, for SP-MWOP the integral GMVS and integral MVS are a bit higher that their fractional version. We believe that this gap will be reduced as the number of services is increased (i.e. as n is increased). We also see that for all three cost functions the dynamic algorithm cost is higher that the fractional version of the GMVS and MVS algorithm costs, yet very close to them. Also, the BL and BS algorithm results for the three cost functions are much higher than those of the dynamic algorithm. The integral GMVS, integral MVS, BL and BS algorithm costs divided by the dynamic algorithm cost for the three cost functions is shown in Figure 2.

To get a feeling of the main loop runtime complexity in both GMVS and MVS algorithms, we calculated the number of main loop iterations (t) in each algorithm run and divided it

[3] Note that we could probably improve results by moving each stick to the closest integral point, which is either left or right of it. However, we think that this improvement is minor when n gets larger.

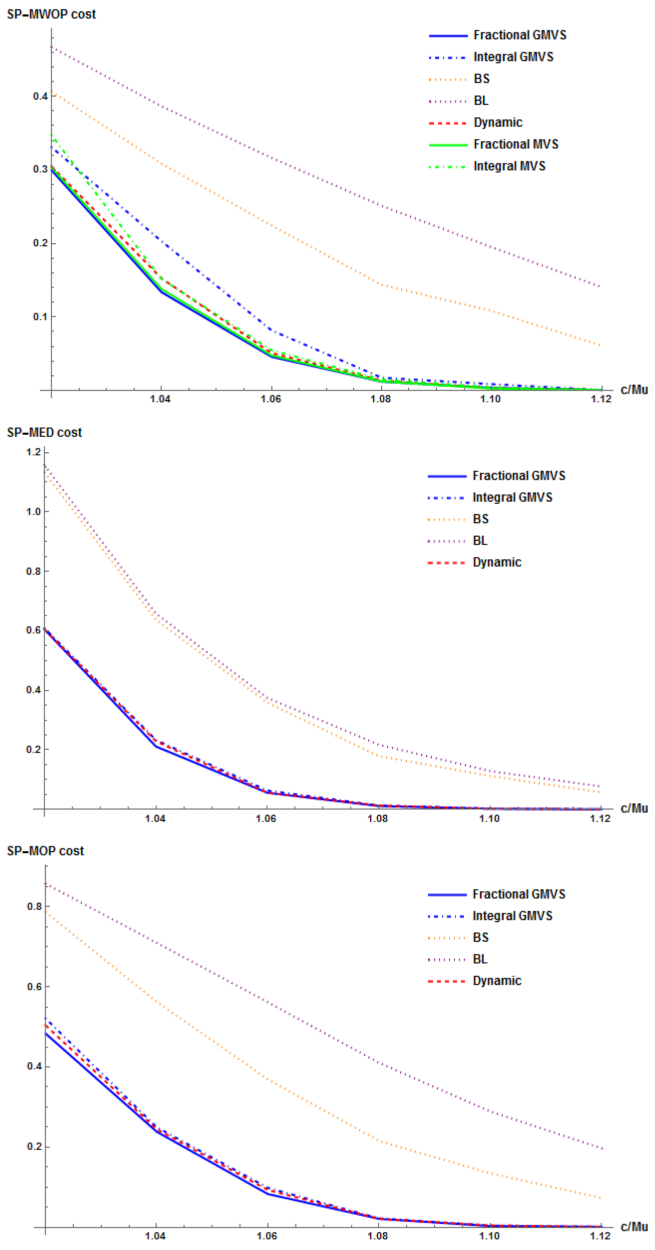

Figure 1 Average cost of the Fractional and Integral GMVS and MVS algorithms, and the BS and BL algorithms for SP-MWOP, SP-MED and SP-MOP with four bins and synthetic normally distributed data. $n = 400$. $\tau = 0.001$. The x axis measures $\frac{c}{\mu}$.

by the number of services (n). The average results are shown in Figure 3. We can see that in all cases the number of iterations dose not surpass $0.6n$. We also see that the number of iterations increases as the spare capacity increases. When the spare capacity increases, the bins are less full. Therefore, the final bin of each service is more likely to be different than (and more distant from) it's initial bin, and hence more service movements are required, i.e., more main loop iterations are required. In any case, the process always terminates quickly.

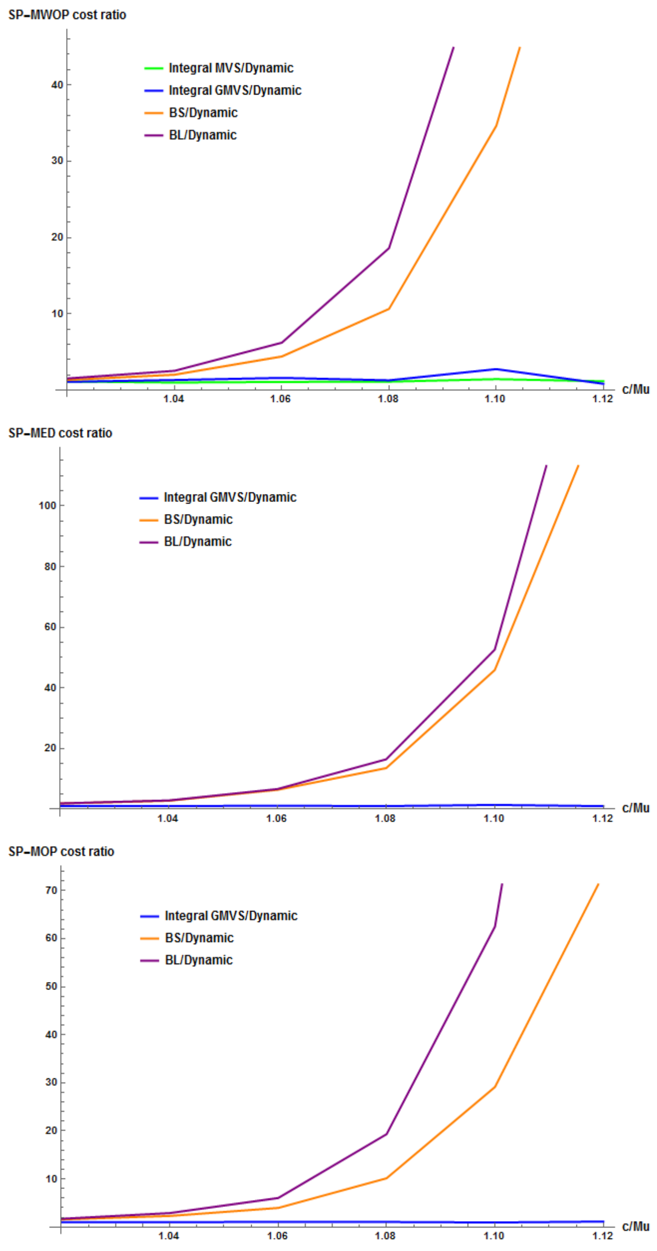

Figure 2 The BL and BS algorithm costs divided by the dynamic algorithm cost for SP-MWOP, SP-MED and SP-MOP with four bins and synthetic normally distributed data. $n = 400$. $\tau = 0.001$. The x axis measures $\frac{c}{\mu}$.

A.2 Results for Real Data

The real data for this part is exactly the one we used for the 2 bin case in [6]. Since the dynamic algorithm run time for $n = 6099$ is too long, we didn't use it for the real data simulations.

In Figure 4 (left) we see that differences between the results of the GMVS and the MVS algorithms (both integral and fractional) are so minor that it is difficult to distinguish the

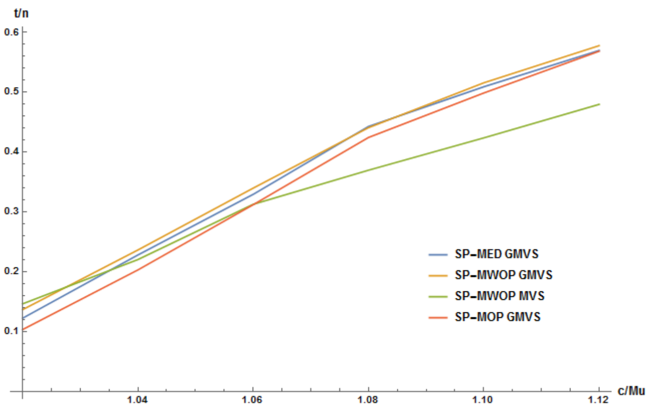

Figure 3 Average main loop number of iterations (t) divided by number of services (n) in the GMVS and MVS algorithms for the various cost functions with four bins and synthetic normally distributed data. $n = 400$. $\tau = 0.001$. The x axis measures $\frac{c}{\mu}$.

lines, thus in the rest of the text we treat GMVS and MVS as one.

Figure 4 shows that for all cost functions considered (SP-MWOP, SP-MED, SP-MOP) the Integral GMVS is very close to the Fractional GMVS, as expected. Also the BL and BS results are much higher than those of Integral GMVS. The BL and BS algorithm costs divided by the Integral GMVS algorithm cost for the three cost functions is shown in Figure 5.

Finally, as in the synthetic data simulations, we calculated the number of main loop iterations (t) in each algorithm run and divided it by the number of services (n). Figure 6 shows that in all cases the average number of iterations does not surpass $1.5n$.

References

1 David Breitgand and Amir Epstein. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In *IEEE INFOCOM 2012*, pages 2861–2865.

2 Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *IEEE FOCS 1999*, pages 579–586.

3 Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.

4 Evdokia Nikolova. Approximation algorithms for offline risk-averse combinatorial optimization. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 338–351, 2010.

5 Evdokia Nikolova, Jonathan A Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *European Symposium on Algorithms*, pages 552–563. Springer, 2006.

6 Galia Shabtai, Danny Raz, and Yuval Shavitt. Risk aware stochastic placement of cloud services: The case of two data centers. *Submitted to ALGOCLOUD 2017 (submission 10).*

7 Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *IEEE INFOCOM 2011*, pages 71–75.

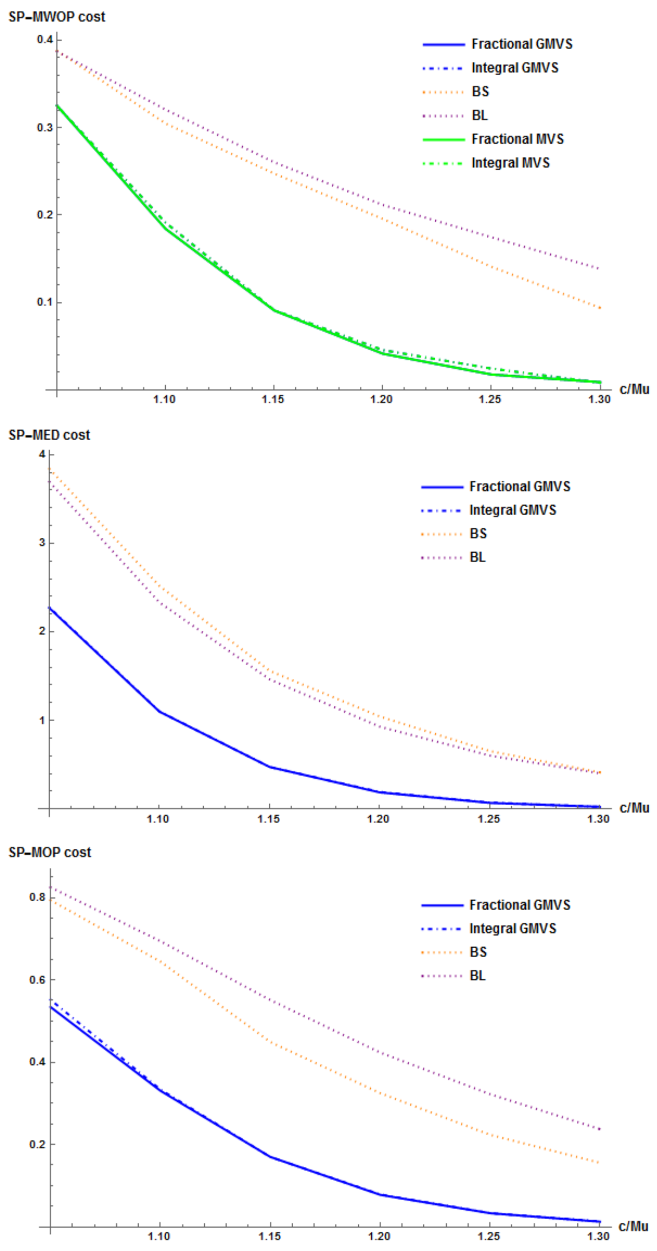

Figure 4 Average cost of the Fractional and Integral GMVS and MVS algorithms, and the BS and BL algorithms for SP-MWOP, SP-MED and SP-MOP with four bins and real independent data. $n = 6099$. $\tau = 0.00001$. The x axis measures $\frac{c}{\mu}$.

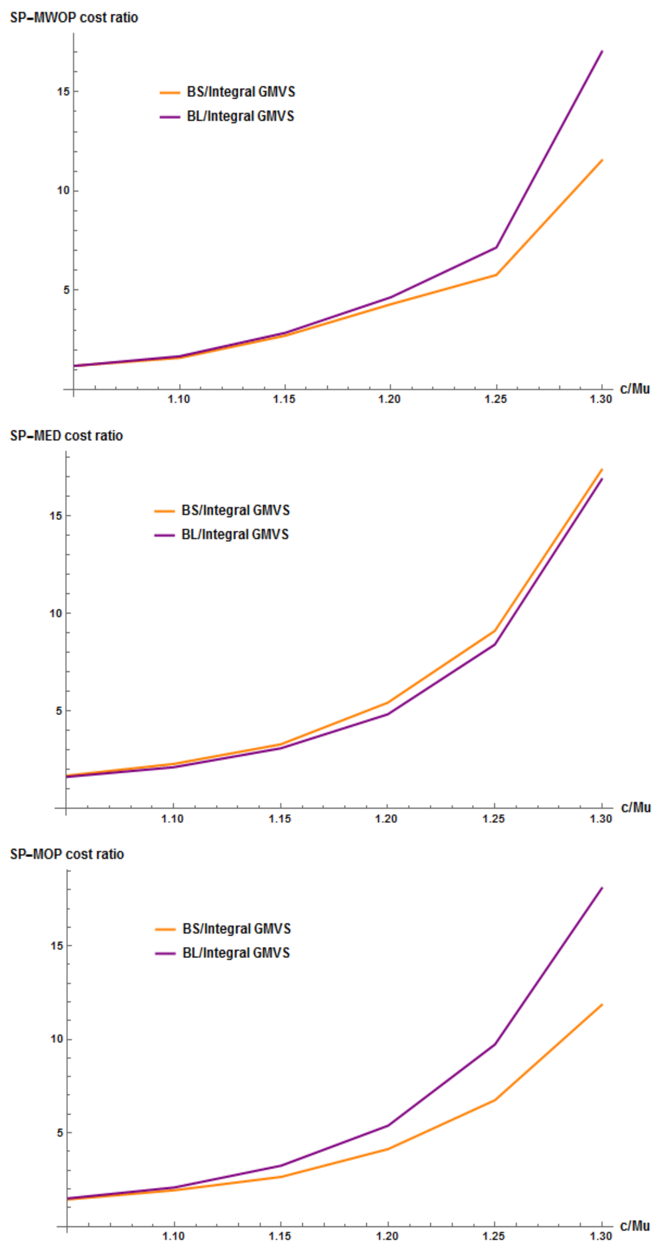

Figure 5 The BL and BS algorithm costs divided by the Integral GMVS algorithm cost for SP-MWOP, SP-MED and SP-MOP with four bins and real independent data. $n = 6099$. $\tau = 0.00001$. The x axis measures $\frac{c}{\mu}$.

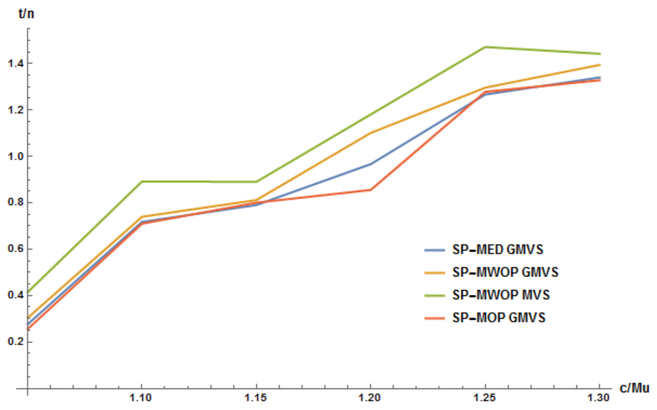

Figure 6 Average main loop number of iterations (t) divided by number of services (n) in the GMVS and MVS algorithms for the various cost functions with four bins and real independent data. $n = 6099$. $\tau = 0.00001$. The x axis measures $\frac{c}{\mu}$.