

Distributed Approximation Algorithm for Resource Clustering

Olivier Beaumont, Nicolas Bonichon, Philippe Duchon, Hubert Larchevêque

Universit de Bordeaux, INRIA Bordeaux Sud-Ouest, Laboratoire Bordelais de Recherche en Informatique

Abstract. In this paper, we consider the clustering of resources on large scale platforms. More precisely, we target parallel applications consisting of independant tasks, where each task is to be processed on a different cluster. In this context, each cluster should be large enough so as to hold and process a task, and the maximal distance between two hosts belonging to the same cluster should be small in order to minimize latencies of intra-cluster communications. This corresponds to maximum bin covering with an extra distance constraint. We describe a distributed approximation algorithm that computes resource clustering with coordinates in \mathbb{Q} in $O(\log^2 n)$ steps and $O(n \log n)$ messages, where n is the overall number of hosts. We prove that this algorithm provides an approximation ratio of $\frac{1}{3}$.

1 Introduction

The past few years have seen the emergence of a new type of high performance computing platform. These highly distributed platforms, such as BOINC [3] or WCG [2] are characterized by their high aggregate computing power and by the dynamism of their topology. Until now, all the applications running on these platforms (seti@home [4], folding@home [1],...) consist in a huge number of independent tasks, and all data necessary to process a task must be stored locally in the processing node. The only data exchanges take place between the master node and the slaves, which strongly restricts the set of applications that can be performed on this platform. Two kind of applications fit in this model. The first one consists in application, such as Seti@home, where a huge set of data can be arbitrarily split into arbitrarily small amount of data that can be processed independently on participating nodes. The other application that are executed on these large scale distributed platforms correspond to Monte-Carlo simulations. In this case, all slaves work on the same data, except a few parameters that drive Monte Carlo simulation. This is for instance the model corresponding to Folding@home. In this paper, our aim is to extend this last set of applications. More precisely, we consider the case where the set of data needed to perform a task is possibly too large to be stored at a single node. In this case, both processing and storage must be distributed on a small set of nodes that will collaborate to perform the task. The nodes involved in the cluster should have an aggregate memory larger than a given threshold, and they should be close

enough (the latencies between those nodes should be small) in order to avoid high communication costs. In this paper, we focus on a preliminary subproblem, namely, that of efficiently forming groups of participating nodes that will be able to collaborate for solving these tasks.

This corresponds, given a set of weighted items (the weights are the storage capacity of each node), and a metric (based on latencies), to create a maximum of groups so that the maximal latency between two hosts inside each group is lower than a given threshold, and so that the total storage capacity of a group is greater than a given storage threshold. This problem turns out to be difficult, even if one node knows the whole topology (i.e. the available memory at each node and the latency between each pair of nodes). Indeed, even without the distance constraint, this problem is equivalent to the classical NP-complete bin covering problem [6]. Similarly, if we remove the constraint about storage capacity, but keep the distance constraint, the problem is equivalent to the NP-Complete disk cover problem [10].

Moreover, in a large scale dynamic environment such as BOINC, where nodes connect and disconnect at a high rate, it is unrealistic to assume that a node knows all platform characteristics. Therefore, in order to build the clusters, we need to rely on fully distributed schemes, where a node makes the decision to join a cluster based on its position, its weight, and the weights and positions of its neighbor nodes. In order to estimate the position of the nodes involved in the computation, we rely on mechanisms such as Vivaldi [7,9] that associate to each node a set of coordinates in a low dimension metric space, so that the distance between two points approximates the latency between corresponding hosts.

To the best of our knowledge, this paper is the first attempt to consider both criteria (memory and latency) simultaneously. Therefore, since our aim is also to design fully distributed algorithms, we limit our study to the case where the coordinates of the points lie in \mathbb{Q} and we consider memory constraints only (one could imagine to add other constraints, such as a minimal aggregate computing power or a minimal aggregate disk storage).

In this paper, we present a fully distributed greedy algorithm that creates a number of bins approximating the optimal within a $\frac{1}{3}$ ratio. This result can be compared to some results on classical bin covering in centralized environment without the distance constraint. In this (easier) context, a *PTAAS* (polynomial-time asymptotic approximation scheme) has been proposed for bin covering [8], i.e. algorithms A_ϵ such that for any $\epsilon > 0$, A_ϵ can perform, in a polynomial time, a $(1 - \epsilon)$ -approximation of the optimal when the number of bins tends towards the infinite. Many other algorithms have been proposed for bin covering, such as [6], that provides algorithms with approximation ratio of $\frac{2}{3}$ or $\frac{3}{4}$, still in a centralized environment.

The $\frac{1}{3}$ approximation ratio algorithm we propose in this paper takes both memory and distance constraints into account and can be efficiently implemented in practice on a Peer To Peer (P2P for short) platform using a skip-graph [5] as overlay. The overall number of exchanged messages is of order $O(n \log n)$ and it takes at most $O(\log^2 n)$ rounds to converge.

The rest of the paper is organized as follow : in Section 2, we prove that any "reasonable" (i.e. satisfying two natural constraints) greedy algorithm leads to an approximation ratio of $\frac{1}{3}$. In Section 3, we present an efficient distributed algorithm to compute prefix sums using a skip graph [5]. This algorithm is later used in Section 4, where we detail the implementation and the theoretical properties of the distributed approximation algorithm we propose for solving bin covering problems with distance constraint.

2 Distance constrained bin covering: greedy approximation

2.1 Bin Covering subject to distance constraints

Since our aim is to build clusters whose aggregate power is large enough and such that any two nodes belonging to the same cluster are close enough, we introduce the "Distance Constrained Bin Covering" decision problem (DCBC for short).

Definition 1 (DCBC: Distance Constrained Bin Covering).

Input: – a set $S = \{s_1, \dots, s_n\}$ of elements,
– a position function $pos : S \rightarrow E$ where (E, d) is a metric space,
– a weight function $w : S \rightarrow \mathbb{Q}^+$,
– a weight threshold W ,
– an integer K ,
– a distance bound d_{max} .

Output: Is there a collection of K pairwise disjoint subsets S_1, \dots, S_K of S such that $\forall i \leq K: \sum_{s \in S_i} w(s) \geq W$ and $\forall (u, v) \in S_i, d(u, v) \leq d_{max}$?

Clearly, DCBC is NP-Complete, since the case where all elements are at the same location corresponds to the classical Bin Covering problem. In what follows, for the sake of clarity, we normalize the weights of the elements (divide them by W) and set $W = 1$ and we do not consider elements whose weight is larger than 1, since such an element can form a group by itself and can be removed. In the rest of the paper, we propose several approximation algorithms (both centralized and decentralized) for the corresponding optimization problem we call *max_DCBC*, in the restriction when the metric space E is taken to be \mathbb{Q} with the usual distance.

2.2 Approximation ratio of $\frac{1}{3}$

In this section, we propose an algorithm that provides an approximation ratio of $\frac{1}{3}$ for *max_DCBC*. We say that a bin is *interval-based* if the positions of all the elements of this bin belong to an interval and if any element whose position is in this interval belongs to the considered bin. On the other hand, we only deal with *greedy* bins, in the sense that the weight of a bin is strictly less than 2,

considering that when the weight of a bin is 2 or more, any element of this bin can be removed without invalidating the bin.

The following Lemma states that any maximal solution consisting of minimal interval-based bins reaches an approximation ratio of $\frac{1}{3}$ with respect to the optimal (*i.e.* without the interval-based condition) solution.

Lemma 1. *A solution to max_DCBC problem that satisfies the following constraints*

P1: *Bins are interval-based and the weight of each bin is strictly smaller than 2,*

P2: *No valid interval-based bin can be created with the remaining elements,*

approaches the optimal solution within a ratio $\frac{1}{3}$.

Note that these are the conditions satisfied by the solutions given by a greedy algorithm that repeatedly creates interval-based bins until none are possible any more.

Proof. Let us consider a solution $\mathcal{S} = \{B_1, \dots, B_k\}$ of *max_DCBC* satisfying the above conditions, and S^* an optimal solution (not necessarily satisfying the above conditions). Let us denote by $l(B_i)$ (resp. $r(B_i)$) the left (resp. right) bound of B_i . We assume that $\forall 1 \leq i \leq k-1, r(B_i) < l(B_{i+1})$, for the sake of simplicity.

We define the *extended area* of B_i as the interval $[l(B_i) - d_{max}, r(B_i) + d_{max}]$.

Clearly, because of P2, any bin in S^* intersects one of the B_i 's and is therefore included in its extended area.

Chains: We call chain a maximal sequence of consecutive bins (B_j, \dots, B_{j+k-1}) such that, as in Figure 1, each extended area of each bin B_i of the sequence intersects the neighbouring bin ($r(B_i) + d_{max} \geq l(B_{i+1})$). Consider the extended area of a chain, denoted as the interval I_c , where $I_c = [l(B_j) - d_{max}; r(B_{j+k-1}) + d_{max}]$ for each chain c of a solution.

Since any bin B in S^* is included in the extended area of a bin of our solution, each bin in S^* is included in the extended area of at least one chain. We arbitrarily affect each bin of S^* to a chain of our solution.

We prove in what follows that no more than $3k$ bins of S^* can be affected to a chain containing k bins, hence the approximation ratio of $\frac{1}{3}$.

We now claim that $w([l(B_j) - d_{max}; r(B_{j+k-1}) + d_{max}]) < 3k + 1$, for a chain $c = (B_j, \dots, B_{j+k-1})$ containing k bins. This obviously implies that no more than $3k$ bins of S^* can be included in I_c . Indeed, the considered interval is the union of k bin intervals (each with weight $w(B_i) < 2$ as per P1), $k-1$ intervals of the form $(r(B_i), l(B_{i+1}))$ (each with weight strictly less than 1 as per P2), and of the two intervals $[l(B_j) - d_{max}, l(B_j))$ and $(r(B_{j+k-1}), r(B_{j+k-1}) + d_{max}]$ (also each with weight strictly less than 1). The total weight in the extended area of a chain containing k bins is $w(c) < 3k + 1$, thus no more than $3k$ bins of S^* can be affected to this chain. Summing over all chains, the number of bins in S^* cannot be larger than three times the number of bins in \mathcal{S} .

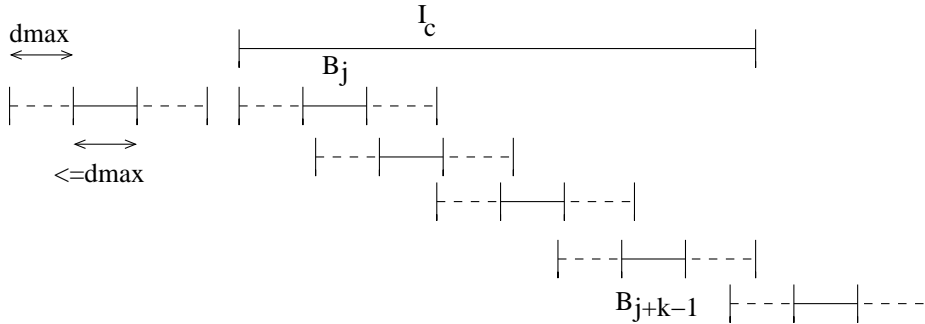


Fig. 1. Example of a chain of 4 bins

3 Preliminaries: overlay and prefix sums computation

Our approximation algorithm for *max_DCBC*, described in Section 4, runs on a special overlay network, and uses prefix sums computations as a preliminary step. In this section, we describe both elements.

3.1 The skip-graph overlay

To each element of the list to group is associated a host of the network.

First we will present the overlay we use for the network, whose goal is to minimize the number of exchanged messages necessary to perform the bin covering of the elements, while being easy to maintain and construct. This overlay is a skip-graph [5], a structure inspired by skip-lists [12,11].

A skip-list is organized as a succession of ordered lists containing fewer and fewer nodes at each level i : for $i > 0$, each host at level $i - 1$ appears in the list of level i with probability p (we use $p = 1/2$). The hosts belonging to the level i list are linked to each other according to the coordinates' order of the basic list.

In the overlay network we use, hosts are organized as nodes of a skip-graph, and ordered by their positions, *i.e.* their coordinates in \mathbb{Q} . To make notations simpler, we identify each host x with its position and write $x \leq y$ for $pos(x) \leq pos(y)$, where pos is the position function as defined in Section 2.

In a skip-graph, there are up to 2^i lists at level i , each indexed by a binary word of length i . Each host x generates a key $key(x)$, a sequence of random bits, using a sequence of random coin flips. At each level i , the level 0 list L_c , containing all hosts ordered by coordinates, is partitioned into 2^i lists, each containing the hosts sharing the same first i key bits. Each host stops its key generation as soon as it is the only host in a level i list, *i.e.* as soon as no other host shares the exact first i key bits. Then it only keeps the first i bits of its key.

For an example, see Figure 2.

Notations Let x be a host. We use $b_i(x)$ to denote the first i bits of $key(x)$, $h(x)$ to denote the number of bits of $key(x)$, and $L_i(x)$ the level i list x belongs to, i.e. $L_i(x) = L_{b_i(x)}$. We also use $\text{Pred}(L, x)$ as the predecessor of x in the list L (or $-\infty$ if x is the first in L), and, likewise, $\text{Succ}(L, x)$ as the successor of x in the list L (or $+\infty$ if x is the last in L). In fact in what follows we more often use $\text{Pred}_i(x)$ to denote $\text{Pred}(L_i(x), x)$ (and $\text{Succ}_i(x)$ for the equivalent successor).

We recall a few facts about the skip-graph data structure, detailed in [5]

Theorem 1 ([5] Search time in a skip-graph). *The expected search time in a skip graph is $O(\log n)$.*

Theorem 2 ([5]). *With high probability, the maximum height of an element in a n -node skip-graph is $O(\log n)$.*

We will use this overlay to compute the prefix sums for each host.

3.2 Prefix sums computation

We recall the fact that each host x has a weight $w(x) < 1$. We define the *prefix sum* $S(x)$ of a host x to be *the sum of the weights of every host in L_ϵ between the first host and x , included*. We also define a *level i partial prefix sum* $S_i(x)$:

$$S_i(x) = \sum_{\substack{z \in L_\epsilon \\ \text{Pred}_i(x) < z \leq x}} w(z). \quad (1)$$

Thus $S_i(x)$ is *the sum of the weights in the basic list, L_ϵ , of every host between its level i predecessor (excluded) and itself (included)*. It can also be defined as $S(x) - S(\text{Pred}_i(x))$.

To compute $S_i(x)$, we use the following recurrence: Thus we have:

$$S_i(x) = \text{sum}_{\substack{z \in L_{i-1}(x) \\ \text{Pred}_i(x) < z \leq x}} S_{i-1}(z). \quad (2)$$

(To prove equation 2, notice that the first z in the summation is $\text{Succ}_{i-1}(\text{Pred}_i(x))$, and the positive and negative terms cancel each other, leaving only $S(x)$ and $-S(\text{Pred}_{i-1}(\text{Succ}_{i-1}(\text{Pred}_i(x)))) = -S(\text{Pred}_i(x))$.)

Note that $S_0(x) = w(x)$ and $S(x) = S_{h(x)}(x)$.

Algorithm Algorithm 1, running on each host, uses the skip-graph to compute the partial prefix sum for each host at each level. It runs upward in the levels of the skip-graph, ending at the highest level for each host. When the computation ends at a host x , it knows each of its partial prefix sums $S_i(x)$.

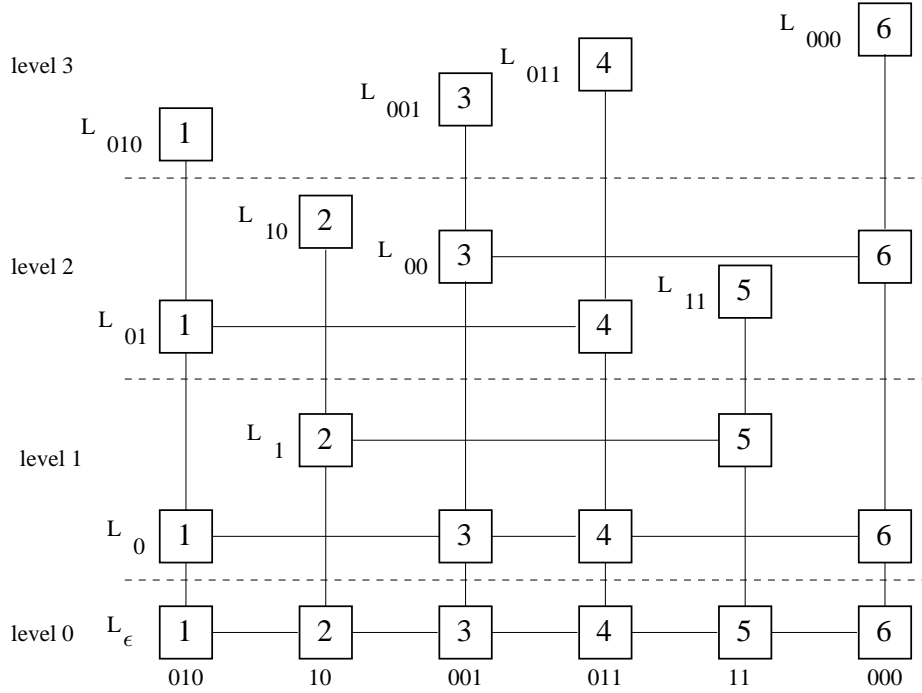


Fig. 2. Example of skip-graph, with 4 levels and 6 nodes in L_ϵ

Time model We consider that each message transmission is done in a unit time, and we do not consider computation time on each host.

Theorem 3. *Algorithm 1 computes all the prefix sums, using a skip-graph as overlay, in $O(\log^2 n)$ time steps and $O(n \log n)$ messages, with high probability.*

Proof. **Number of messages**

- Each message has this form: $\langle i, b, w \rangle$. We call i the message level, b the message key and w the message weight.
- Each message weight is null when it is first emitted (before the first retransmission).
- Each level i message is sent by a host to its level i successor.
- Whenever a level i message is received by a host x with a key not matching $b_i(x)$, the message sent in response (line 10 of Algorithm 1) is considered to be the same message, retransmitted with increased weight (this still counts

Algorithm 1 Algorithm for a host x

```
1:  $S_0(x) = 0$ 
2: for each level  $i = 0, 1, 2, \dots$  do
3:   if  $\text{Pred}_i(x) == -\infty$  then
4:     send to  $\text{Succ}_i(x)$ :  $\langle i, b_{i+1}(x), 0 \rangle, \langle i, b_{i+1}(x), S_i(x) \rangle$ 
5:   else if  $\text{Succ}_i(x) \neq +\infty$  then
6:     send to  $\text{Succ}_i(x)$ :  $\langle i, b_{i+1}(x), 0 \rangle$ 
7:     for each received message  $\langle i, b, w \rangle$  do
8:       as soon as  $S_i(x)$  is known:
9:       if  $b \neq b_{i+1}(x)$  then
10:        send  $\langle i, b, w + S_i(x) \rangle$  to  $\text{Succ}_i(x)$ 
11:       else
12:         $S_{i+1}(x) = S_i(x) + w$ 
13:       end if
14:     end for
15:   end if
16: end for
```

in the total number of messages). Note that the first host x of each level i list sends automatically two messages, so that the first host to its right with a key not matching $b_i(x)$ also computes its S_{i+1} sum on line 12.

- As soon as a level i message is received by a host x with a key matching $b_i(x)$, it is not retransmitted anymore (line 12 of Algorithm 1), and its weight is used by the receiving host to compute its level $i + 1$ partial prefix sum.

Note that, given a list at level i , and two neighbouring hosts in this list, only two messages are exchanged between those two hosts at this level, sent by the leftmost one to the rightmost one.

In Algorithm 1, each host sends two messages by level: at a given level, the first host of the list sends its two messages on line 4, and every other host sends one message on line 6, and the other on line 10. Thus each host x send $2h(x)$ messages (and receive as many messages). By Theorem 2, with high probability, $O(n \log n)$ messages are sent during the execution of Algorithm 1.

Time analysis

If $x_1 \dots x_k$ are consecutive elements of a same level i list L , such that all belong to the same level $i + 1$ list L' , and $\text{Pred}(L, x_1) \notin L'$, and $\text{Succ}(L, x_k) \notin L'$, we call $\{x_1, \dots, x_k\}$ a level i siblings set. Note that at each level, the largest siblings set is of length $O(\log n)$ with high probability. For more information on this result, see [12].

Each level i message is transmitted through a whole siblings set before it reaches its destination y . During these transmissions, it collects the partial prefix sums contributions to $S_i(y)$. When a host receives a message at a time T , it can retransmit it at a time $T + 1$.

If we write $T_i(x)$ for the instant where x learns $S_i(x)$, $T_i = \max_{x \in L_\epsilon} T_i(x)$, and M for the maximal size of all the siblings sets:

$$T_i(x) = \max_{j=0..m} (T_{i-1}(y_j) + m - j + 1) \quad (3)$$

where $y_0 = \text{Pred}_i(x)$, m is the size of the (potentially empty) level $i - 1$ siblings set between $\text{Pred}_i(x)$ and x , and $y_j, 1 \leq j \leq m$ is the j^{th} host of this siblings set. Note that for all hosts x , $T_0(x) = 0$.

Thus

$$T_i(x) \leq \max_{j=0..m} (T_{i-1}(y_j) + m + 1) \leq T_{i-1} + M + 1. \quad (4)$$

Thus $T_i \leq T_{i-1} + M + 1$. Thus, if we write $H = \max_{x \in L_\epsilon} h(x)$, we have $T_H \leq H(1 + M)$. Both H and M are $O(\log n)$; the upper bound on the number of time steps follows.

Note that we suspect the $O(\log^2 n)$ bound is pessimistic, and that a tighter analysis would yield $\Theta(\log n)$.

4 A distributed approximation algorithm

In this section, the n hosts to cluster are linked by a skip-graph as overlay network. Hosts are identified to their index in L_ϵ , *i.e.* we write $i \leq j$ if the host i is before j in L_ϵ . Sending a message from a host i to the host j using the overlay network can be done with $O(\log |i - j|)$ messages with high probability.

First we present a distributed algorithm computing the clustering of a list of weighted hosts, without the distance restriction. Then this restriction is added to our algorithm in subsection 4.2. The final algorithm has an approximation ratio of $\frac{1}{3}$, as per Lemma 1.

Note that any greedy algorithm without distance restriction has an obvious approximation ratio of $\frac{1}{2}$, if it creates groups weighting strictly less than 2.

The following algorithms creates interval-based bins in L_ϵ . Each created bin corresponds to a cluster of the platform. The clusters are defined in the following way: each grouped host received the identifier of a host, called *leader*. A cluster is a set of hosts having the same leader. Here the leader of a cluster is always the leftmost host of the group.

4.1 Clustering Algorithm without distance constraints

The recursive treatment is based on a procedure that takes 3 parameters: li (left index), ri (right index) and rli (right leader index), with $li < rli \leq ri$. This routine is executed by host $j = \lfloor \frac{li+ri}{2} \rfloor$. This procedure creates clusters included in the intervals $[li, ri)$, having their leader in the intervals $[li, rli[$. To begin the clustering of the hosts, send $\langle 0, n, n \rangle$ to host $\lfloor n/2 \rfloor$. The principle of this routine is quite simple. If host j can be leader of a cluster included in interval $[li, ri)$, it creates this cluster and recursively call the procedure on the

two sublists respectively before and after the newly created cluster. If it cannot be leader, no other host after j can create such a cluster in interval (j, rli) , hence it will only look for leaders in the sub-interval $[li, j - 1)$ for clusters included in $[li, ri)$.

The recursive routine is described in Algorithm 2. It uses two sub-routines: $rb(j)$ and $buildCluster(a, b)$. The function $rb(j)$ returns the minimum host k after j such that $p_k \geq 1 + p_j$, where p_j is the prefix sum of host j ; *i.e.* the leftmost host that could possibly be the last host in an interval-based bin having j as its first host. By convention, $rb(j) = +\infty$ if no such host exists. The routine $buildCluster(a, b)$ defines a as leader for each host $a \leq j \leq b$.

Algorithm 2 Algorithm for a coordinator j

```

1: Receive  $\langle li, rli, ri \rangle$ 
2: if  $j < ri$  then
3:   if  $rb(j) < ri$  then
4:     buildCluster( $j, rb(j)$ )
5:     Send  $\langle li, j, j \rangle$  to host  $\lfloor \frac{li+j}{2} \rfloor$ 
6:     if  $rb(j) < rli$  then
7:       Send  $\langle rb(j) + 1, rli, ri \rangle$  to host  $\lfloor \frac{rb(j)+rli+1}{2} \rfloor$ 
8:     end if
9:   else
10:    Send  $\langle li, j, ri \rangle$  to host  $\lfloor \frac{li+j}{2} \rfloor$ 
11:   end if
12: end if

```

Theorem 4. Consider n hosts linked by a skip-graph as overlay network. Algorithm 2 computes a clustering of those hosts such that: (i) each cluster is an interval of L_ϵ of total weight $w < 2$; (ii) each interval of L_ϵ of non-grouped hosts has a weight $w < 1$. Moreover, Algorithm 2 runs in $O(\log^2 n)$ steps and $O(n \log n)$ messages with high probability.

Proof. Clustering: A host is said to be *coordinator* of an interval $[li; ri)$ when it receives a message $\langle li, rli, ri \rangle$ and begins to execute Algorithm 2. It is coordinator of this interval until it begins to send messages on line 5 or on line 10. Thus, at each instant, for each host j , at most one host is the coordinator of an interval containing j .

Each coordinator creates, if it can, a cluster *included* in the interval of which it is coordinator. Then it designates at most two hosts as coordinators of sub-intervals not containing any host of the created cluster. So for two clusters having hosts $(j, k) \in [0; n)$ as leaders, $[j; rb(j)) \cap [k; rb(k)) = \emptyset$. Thus any host affected to a cluster can not be affected later to another cluster: a built cluster is definitively built. By definition of $rb(j)$, constructed clusters have a weight $w < 2$. (i) is thus proved.

To prove (ii), notice that there is a bijection between the set of coordinator hosts and the set of maximum intervals of non-grouped hosts. At the beginning, one host is coordinator of the whole interval of non-grouped hosts. Then at each interval subdivision, either a cluster is created (line 4), and two hosts are designated as coordinators of the two sub-intervals of non-grouped hosts at each side of the created cluster (these two sub-intervals are still maximal in terms of non-grouped hosts); or no cluster is created (line 9) and another host is designated coordinator of the same maximal interval of non-grouped hosts.

Thus at the end of the execution of the algorithm, each maximal interval $[li; ri]$ of non-grouped hosts has host li as coordinator. Since this coordinator does not create a cluster, $rb(li) \geq ri$. Hence the cumulated weight of this maximal interval is strictly less than 1. (ii) is thus proved.

Complexity: In the precomputation steps, the value $rb(j)$ is computed by each host j . This can be done in two steps: first compute the prefix sum for each host (see Section 3.2). Then, based on the prefix sum, search for host $rb(j)$. By Theorem 1, this search in a skip-graph takes for each host a $O(\log n)$ time w.h.p..

The routine $\text{buildCluster}(a, b)$ takes $O(h \log h)$ messages and $O(\log h)$ steps with $h = b - a$. Since all built bins are pairwise disjoint intervals, all execution of buildCluster can be executed in parallel and will take $O(\log h_{max})$ time steps and $O(n \log n)$ messages total, w.h.p..

Each time a host executes Algorithm 2, it designates two hosts to be coordinators of two sub-intervals (lines 5 and 7), or one host to be coordinator of half the original one (line 10). Hence there are at most $\log n$ levels of recursion. As each call of Algorithm 2 takes $O(\log(rli - li))$ messages w.h.p., it takes $O(n \log n)$ messages and $O(\log^2 n)$ steps total w.h.p..

Corollary 1 (Approximation ratio). *The resulting clustering of Algorithm 2 has an approximation ratio of 1/3 with the optimal solution.*

Proof. To prove this corollary, consider Lemma 1, using $d_{max} = \max_{(u,v) \in L_\epsilon} (d(u, v))$. The first property is proved in Theorem 4, and the third property of Lemma 1 is obviously verified as all hosts are at distance less than d_{max} from each other.

4.2 Adding the distance restriction

We recall that each host j has a position $pos(j)$, and that hosts are ordered in L_ϵ by their position.

A host j is *eligible* if it can be the leader of a cluster, that is if $pos(rb(j)) \leq 1 + pos(j)$. The function $neh(j)$ return the *next eligible host* k after j . By convention, $neh(j) = +\infty$ if no such host exists. As for the previous algorithm, $neh(j)$ and $rb(neh(j))$ can be precomputed for each host j . Algorithm 3 is an adaptation of Algorithm 2 that produces clusters of bounded diameter.

Algorithm 3 Algorithm for a coordinator j

```
1: Receive  $\langle li, rli, ri \rangle$ 
2: if  $li < ri$  then
3:   if  $rb(neh(j)) < ri$  then
4:     buildCluster( $neh(j)$ ,  $rb(neh(j))$ )
5:     Send  $\langle li, j, neh(j) \rangle$  to host  $\lfloor \frac{li+j}{2} \rfloor$ 
6:   if  $rb(neh(j)) + 1 < rli$  then
7:     Send  $\langle rb(neh(j)) + 1, rli, ri \rangle$  to host  $\lfloor \frac{rb(neh(j))+rli+1}{2} \rfloor$ 
8:   end if
9:   else
10:    Send  $\langle li, j, ri \rangle$  to host  $\lfloor \frac{li+j}{2} \rfloor$ 
11:  end if
12: end if
```

Theorem 5. Consider n hosts with given positions, linked by a skip-graph as overlay network respecting the order of their positions. Algorithm 3 computes a clustering of those hosts with distance constrained such that: (i) each cluster is an interval of L_ϵ of diameter at most d_{max} and of weight $w < 2$; (ii) each interval of L_ϵ of non-grouped hosts of diameter $d \leq d_{max}$ has a total weight $w < 1$.

Moreover, Algorithm 3 runs in $O(\log^2 n)$ steps and $O(n \log n)$ messages with high probability.

Proof. The complexity of this algorithm is clearly the same as Algorithm 2: at each execution of Algorithm 3, the interval $[li; rli]$ is split in at most two new sub-intervals, each of size at most half the size of the original interval $[li; rli]$.

(i): By definition of $rb(j)$, and with a reasoning similar to the proof of Theorem 2, constructed clusters have a weight strictly lower than 2. Moreover, by definition of $neh(j)$, created clusters have a diameter lower than $d_{max} (\leq d_{max})$.

(ii): The reasoning is still similar to the proof of Theorem 4. At the end of the algorithm the coordinator of $[li; ri]$ is li . As li does not create a cluster we deduce that $rb(neh(li)) \geq ri$, i.e the first host able to build a cluster at the right of li would build a cluster overtaking the right bound ri . By definition of rb and neh we deduce that no cluster of bounded diameter can be created in $[li; ri]$.

Corollary 2 (Approximation ratio). The resulting clustering of Algorithm 2 has an approximation ratio of $1/3$ with the optimal solution.

Proof. The proof of this corollary is a direct application of Lemma 1, of which the two necessary properties are verified, proved in Theorem 5.

5 Conclusions

In this paper we have presented a distributed approximation algorithm, running in $O(\log^2 n)$ steps and $O(n \log n)$ messages, that computes resource clustering

for n hosts with coordinates in \mathbb{Q} . This algorithm provides an approximation ratio of $\frac{1}{3}$. We have restricted this work to a 1-dimensional case, but we are working on the extension of our results to higher dimensions or more general metric spaces.

As this work is meant to be used on large-scale platforms, it is necessary to make our algorithms able to handle a high degree of dynamicity on the hosts of the networks. Notably, they have to handle the dynamicity of the weights of the hosts, because each user of a large-scale platform is likely to want its resources back for its private use at any time.

It could also be interesting to work on a version in which each created cluster would have many criteria to satisfy. In fact in our work, we just considered that clusters had to offer a sufficient global storage capacity, but one may want clusters to additionally ensure, for example, sufficient computing power.

References

1. Folding@home. <http://folding.stanford.edu/>.
2. World community grid. <http://www.worldcommunitygrid.org>.
3. David P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.
4. David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.
5. J. Aspnes and G. Shah. Skip graphs. *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393, 2003.
6. S.F. Assmann, D.S. Johnson, D.J. Kleitman, and J.Y.T. Leung. On a dual version of the one-dimensional bin packing problem. *Journal of algorithms(Print)*, 5(4):502–525, 1984.
7. R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical, distributed network coordinates. *ACM SIGCOMM Computer Communication Review*, 34(1):113–118, 2004.
8. J. Csirik, D.S. Johnson, and C. Kenyon. Better approximation algorithms for bin covering. *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 557–566, 2001.
9. F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, 2004.
10. M. Franceschetti, M. Cook, and J. Bruck. A geometric theorem for approximate disk covering algorithms, 2001.
11. J.I. Munro, T. Papadakis, and R. Sedgewick. Deterministic skip lists. *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 367–375, 1992.
12. William Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Workshop on Algorithms and Data Structures*, pages 437–449, 1989.