A Distributed Algorithm for Resource Clustering in Large Scale Platforms

Olivier Beaumont, Nicolas Bonichon, Philippe Duchon, Lionel Eyraud-Dubois, Hubert Larchevêque

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Distributed Algorithm for Resource Clustering in Large Scale Platforms

Olivier Beaumont — Nicolas Bonichon — Philippe Duchon — Lionel Eyraud-Dubois — Hubert Larchevêque

N° 6883

Octobre 2008

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6883--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
BORDEAUX - SUD OUEST

A Distributed Algorithm for Resource Clustering in Large Scale Platforms

Olivier Beaumont, Nicolas Bonichon, Philippe Duchon , Lionel Eyraud-Dubois , Hubert Larchevêque *

Thème COM — Systèmes communicants
Équipe-Projet CEPAGE

Rapport de recherche n° 6883 — Octobre 2008 — 16 pages

Abstract: We consider the resource clustering problem in large scale distributed platforms, such as BOINC, WCG or Folding@home. In this context, applications mostly consist in a huge set of independent tasks, with the additional constraint that each task should be executed on a single computing resource. We aim at removing this last constraint, by allowing a task to be executed on a (small) set of resources. Indeed, for problems involving large data sets, very few resources may be able to store the data associated to a task, and therefore may be able to participate in the computations. Our goal is to propose a distributed algorithm for a large set of resources that enables to build clusters, each of which will be responsible for processing a task and storing associated data. From an algorithmic point of view, this corresponds to a bin covering problem with an additional distance constraint. Each resource is associated to a weight (its capacity) and a position in a metric space (its location, based on network coordinates such as those obtained with Vivaldi), and the aim is to build a maximal number of clusters, such that the aggregated power of each cluster (the sum of the weights of its resources) is large enough and such that the distance between two resources belonging to the same cluster is kept small (in order to minimize intra-cluster communication latencies). In this paper, we describe a generic 2-phases algorithm, based on resource augmentation and whose approximation ratio is 1/3. We also propose a distributed version of this algorithm when the metric space is $\mathbb{Q}^D$ (for a small value of D) and the L_∞ norm is used to define distances. This algorithm takes $O((4^D)\log^2 n)$ rounds and $O((4^D)n\log n)$ messages both in expectation and with high probability, where n is the total number of hosts.

Key-words: distributed algorithms, resource clustering, approximation algorithms, resource augmentation, P2P overlays, distributed cooperative computing, peer to peer systems, overlay networks

* Université de Bordeaux, Laboratoire Bordelais de Recherche en Informatique

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération 33405 Talence Cedex
Téléphone : +33 5 40 00 69 00

Un Algorithme Distribué pour de l'Agrégation de Ressources sur des Plates-Formes à grande échelle

Résumé : Nous tudions dans cet article le problme de l'agrgation de ressources de faon distribue sur des plateformes grande chelle, comme BOINC, WCG ou Folding@home. Dans ce contexte, les applications consistent principalement en un ensemble important de tches indpendantes, avec la contrainte que chaque tche doit tre excute sur une seule entit de calcul. Notre objectif est de supprimer cette contrainte, en permettant qu'un tche puisse tre excute sur un (petit) ensemble de ressources. En effet, pour des problmes impliquant de gros volumes de donnes, trs peu d'entits de calculs sont susceptibles d'tre capables de stocker les donnes associes une tche, et donc de participer au calcul global. Notre but est de proposer un algorithme distribu qui, tant donn un ensemble de ressources, permettent la construction de groupes de calculs, de faon que chaque groupe puisse tre responsable de l'excution d'une tche et du stockage des donnes associes. D'un point de vue algorithmique, ceci correspond un problme de bin covering avec une contrainte additionnelle de distance. Chaque ressource est associe un poids(sa capacit de stockage) et une position dans un espace mtrique (sa position gographique, base sur des coordonnes rseaux comme celles obtenues grce Vivaldi), et l'objectif est de construire un maximum de groupes de calculs (clusters), tels que la puissance agrge de chacun de ces clusters (la somme des poids des ressources qu'il contient) soit suffisante et que la distance entre deux membres d'un mme groupe reste petite (de faon minimiser les communications l'intrieur d'un mme cluster). Dans cet article, nous dcrivons un algorithme gnrique en 2 phases, bas sur une augmentation de ressources, et fournissant un ratio d'approximation de 1/3. Nous proposons aussi une version distribue de cet algorithme en se plaant dans $\mathbb{Q}^D$ (pour une valeur de D petite) munie de la norme L_∞ pour dfinir les distances. Cet algorithme prend $O((4^D)\log^2 n)$ tapes et ncessite $O((4^D)n \log n)$ messages, la fois en esprance et avec forte probabilit, o n est le nombre total d'htes sur la plateforme.

Mots-clés : algorithmes distribués, agrégation de ressources, algorithmes d'approximation, augmentation de ressources, systmes Pair Pair, calculs coopratifs distribus

1 Introduction

The past few years have seen the emergence of a new type of high performance computing platforms. These highly distributed platforms, such as BOINC [3], Folding@home [1] and WCG [2] are characterized by their high aggregate computing power, their heterogeneity in terms of resource performances and by the dynamism of their topology, due to node arrivals and departures. 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. This strongly restricts the set of applications that can be run on these platforms.

Two kinds of applications fit in this model. The first one consists in those, such as Seti@home, where a huge set of data can be arbitrarily split into arbitrarily small amounts that can be processed independently on participating nodes. The second one corresponds to Monte-Carlo simulations. In this case, all slaves work on the same data, except a few parameters that drive the 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 data set needed to perform a task is possibly too large to be stored at a single node. This situation is very likely to occur in large scale platforms based on the aggregation of strongly heterogeneous resources. 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 capacity (memory, processing power,...) higher than a given threshold, and they should be close enough (the latencies between those nodes should be small) in order to avoid high communication latencies.

Thus the aim is, 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 number 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 the constraint about storage capacity is removed, but the distance constraint is kept, the problem is equivalent to the NP-Complete disk cover problem [13].

In this paper, we propose a generic greedy 2-phases algorithm, based on resource augmentation and whose approximation ratio is $\frac{1}{3}$. More precisely, we use resource augmentation in the following way. We compare the number of clusters (or bins) created by our algorithm with diameter constraint d to the optimal number of bins that could be created with distance $d_{\max}$, where $d > d_{\max}$. This resource augmentation is both efficient and realistic. Indeed, if the aggregated power of the cluster should be larger than a given threshold in order to be able to process the task, the threshold on the maximal latency between two nodes belonging to the same cluster is weaker, and mostly states that nodes belonging to the same cluster should not be too far from each other. Moreover, this resource augmentation enables us to prove a constant approximation ratio ($\frac{1}{3}$) whereas

approximation ratio without resource augmentation would be exponential in the dimension of the metric space (see Section 2.2). We also give an extension of the generic 2-phases greedy algorithm with approximation ratio $\frac{2}{5}$ with the same resource augmentation. These results are to be compared to classical results for bin covering in centralized environment without the distance constraint. In this (much easier) context, a $PTAAS$ (polynomial-time asymptotic approximation scheme) has been proposed for bin covering [9], *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 goes to infinity. 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.

This paper is a follow-up to [7], where the case of a one-dimensional metric space is considered. In order to estimate the positions of the nodes involved in the large scale platform, we rely on mechanisms such as Vivaldi [8, 11] 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. Here, we consider the case where resource locations are given by their coordinates in a metric space $\mathbb{Q}^D$ with arbitrary dimension D. Moreover, in a large scale dynamic environment such as BOINC, where nodes connect and disconnect with a high churn, 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. Therefore, we also propose a distributed version of this algorithm when the metric space is $\mathbb{Q}^D$ (for a small value of D) and the infinity norm is used to define distances. This algorithm takes $O((4^D)\log^2 n)$ rounds and $O((4^D)n\log n)$ messages both in expectation and with high probability, where n is the total number of hosts. Moreover, we claim that this algorithm can be used in practice, since its implementation only relies on classical distributed data structures, such as skip graphs [5].

The rest of this paper is organized as follows. In Section 2, we present in details the problem modeling and we prove in Section 2.2 that a very simple 2-phases greedy algorithm reaches an approximation ration of $\frac{1}{3}$ using resource augmentation. The distributed version of this algorithm (when the metric space is $\mathbb{Q}^D$) is presented in Section 3 and its complexity (in terms of number of rounds and number of messages) is analyzed in Section 3.3. Finally, we give some conclusions and future works in Section 4.

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, we introduce the "Distance Constrained Bin Covering" decision problem (DCBC for short).

Definition 2.1 (DCBC: Distance Constrained Bin Covering)**. Input***: A set $S = \{e_1, \dots e_n\}$ of elements, a position function $p : S \to E$ where (E, d)*

is a metric space, a weight function $w : S \to \mathbb{Q}^+$, a weight threshold W, an integer K and a distance bound d_{max}.

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

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 elements can form bins by themselves and can be removed form the initial instance. In the rest of the paper, we present an approximation algorithm with resource augmentation for the corresponding optimization problem *max_DCBC*. We then provide a distributed algorithm that works when the underlying metric space is $\mathbb{Q}^D$.

Notations Let $w(e)$ the weight of an element e. This notation is extended to the weight of a bin B where $w(B) = \sum_{e \in B} w(e)$ is the weight of B, and to sets of disjoint bins [1]: $w(\mathcal{K}) = \sum_{B \in \mathcal{K}} w(B)$. In what follows, we identify an element e with its position $p(e)$ and *vice versa*.

2.2 Approximation algorithm with resource augmentation

Principle In the resource augmentation model [10, 12], one compares the performance of a particular algorithm $\mathcal{A}$ to that of the optimal (denoted by OPT) in an unfair way. In this paper, the optimal algorithm is restrained to create bins with diameter at most d_{max}, where $\mathcal{A}$ is allowed to create bins of diameter at most $d + 2d_{max}$, with $d \geq d_{max}$. The goal is still to maximize the number of bins created.

Need for resource augmentation Since our goal is to propose a fully distributed algorithm for building clusters, that can be implemented in practice, we need to rely on algorithms based on local properties without backtracking. Therefore, we will concentrate on greedy algorithms only.

To justify the need for resource augmentation, let us consider the following example (see Figure 1). In the case of $\mathbb{Q}^D$, if L_∞ norm is used, a greedy algorithm could build a bin intersecting $3^D - 1$ optimal bins without being able to use any remaining item to build another bin. In this case, one would obtain an exponentially low approximation ratio $\frac{1}{3^D - 1}$. The question of the inapproximability of max_DCBC will be studied in future works.

A greedy algorithm with resource augmentation For bin covering without distance constraints, there is a well-known greedy algorithm with an approximation ratio of $\frac{1}{2}$. In this algorithm, items are successively added to a bin (in an arbitrary order) until the weight of the bin reaches 1. Then, a new bin is opened and filled using the same algorithm with the remaining items. Since any

[1] We will rather use bin when dealing with complexity issues and approximation algorithms, since it corresponds to the term used in the literature, and use cluster when dealing with the application to resource clustering in large scale systems, although both are equivalent.

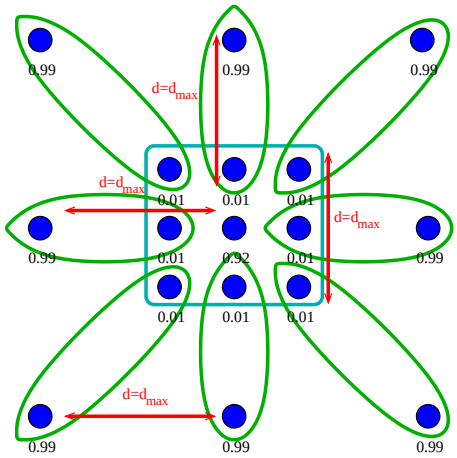


Figure 1: An optimal solution can build 8 bins, whereas a greedy solution may only build 1 bin

item has weight less than 1, the weight of any closed bin is less than 2, which ensures the approximation ratio of $\frac{1}{2}$.

We will now describe a generalization of this algorithm to the distance constrained case, for which we prove a constant approximation ratio, with constant resource augmentation.

The basic structure of this algorithm is made of two phases.

Phase 1 Greedily create bins of diameter at most $d_{\max}$, and stop when it is not possible to create any more such bins.

Phase 2 Greedily create bins of diameter at most $3d_{\max}$.

The intuition behind the second phase comes from the previous example (see Figure 1). We want to ensure that all items that are close to a bin created during Phase 1 are considered during Phase 2. The approximation ratio is actually still valid if, during Phase 2, we only create bins included in the *extended area* (set of points at distance at most $d_{\max}$) of a Phase 1 bin.

Before proving this algorithm, note that even though the stopping criterion for Phase 1 might be difficult to check in a distributed way, it is crucial for the proof of the approximation factor. In order to easily distribute the algorithm, we will consider a (weaker) generalization. For a fixed $d \geq d_{\max}$, bins with diameter at most d will be created in Phase 1, and bins with diameter at most $d + 2d_{\max}$ will be created in Phase 2. The stopping criterion for Phase 1, however, remains the same and Phase 1 stops when it is not possible to create a bin of diameter at most $d_{\max}$. In $\mathbb{Q}^D$, it is possible to cover all sets of diameter $d_{\max}$ with a finite number of candidate sets of diameter $d > d_{\max}$, and this property will be used in the distributed version of the algorithm.

In what follows, we first prove that the greedy algorithm has an approximation ratio of $\frac{1}{3}$. Then, we show that if we can ensure that all created bins during Phase 1 are *thrifty* (i.e. if any element of a bin is removed, then the weight is not large enough), then the approximation ratio becomes $\frac{2}{5}$. Unfortunately,

building *thrifty* bins in a distributed way turns out to be very expensive, and the distributed algorithm we propose in Section 3 has approximation ratio $\frac{1}{3}$.

Lemma 2.1. *The 2-phases greedy algorithm provides a $\frac{1}{3}$-approximation algorithm of* max_DCBC*, using a resource augmentation of factor $2 + \frac{d}{d_{\max}}$ on the maximal diameter of a bin.*

Proof. We define the *extended area* of a bin B to be the set of elements $e \in S$ such that $d(e; B) \leq d_{max}$. Clearly, because of the stopping criterion, in any bin collection, any bin intersects at least one bin created during Phase 1, and is therefore included in its extended area.

Let OPT be an optimal and thrifty solution, *i.e.* one for which in each bin of weight $1+w$, elements have weight less than w. Such a solution always exists since we can modify any optimal solution to ensure this property by removing elements that do not satisfy this property.

We will use the following notations throughout this paper.

- The set of bins created during Phase 1 of the greedy algorithm will be denoted as $\mathcal{K}$, where $|\mathcal{K}| = k$.
- The set of bins created during Phase 2 of the greedy algorithm will be denoted as $\mathcal{M}$, where $|\mathcal{M}| = m$.

Let us split the bins B_i, $1 \leq i \leq k$ created during Phase 1 into three subsets $\mathcal{K}_k$, $1 \leq k \leq 3$, depending on the number of elements $|B_i|$ they contain. Let us denote by:

- $\mathcal{K}_1 = \{B_i \in \mathcal{K} / \ |B_i| = 1\}$ and $k_1 = |\mathcal{K}_1|$ and OPT_1 the set of bins of OPT that intersect bins in $\mathcal{K}_1$,
- $\mathcal{K}_2 = \{B_i \in \mathcal{K} / \ |B_i| = 2\}$ and $k_2 = |\mathcal{K}_2| = k_2$ and OPT_2 the set of bins of OPT that intersect bins in $\mathcal{K}_2$,
- $\mathcal{K}_3 = \mathcal{K} \backslash (\mathcal{K}_1 \cup \mathcal{K}_2)$, and $k_3 = k - k_1 - k_2$ and OPT_3 the set of bins of OPT that intersect bins in $\mathcal{K}_3$.

Let us denote by $|OPT|$ the number of bins created by OPT. Since any bin of OPT intersects a bin built during Phase 1, $|OPT| \leq |OPT_1| + |OPT_2| + |OPT_3|$.

Let us determine more precisely the cardinalities of the sets OPT_1 and OPT_2:

$B \in OPT_1$ If a bin created by our greedy algorithm has exactly one element e, then $w(e) \geq 1$. Since, as stated earlier (see Section 2.1), we do not consider such elements, $OPT_1 = \emptyset$, thus $\mathcal{K}_1 = \emptyset$, $k_1 = 0$.

$B \in OPT_2$ Each bin in $\mathcal{K}_2$ is made of exactly 2 elements. Since bins created by OPT are disjoint, then $|OPT_2| \leq 2k_2$.

To bound the cardinality of OPT_3, let us introduce w_{lost}, the total weight of elements that do not belong to any bin created by the greedy algorithm, and belonging to OPT_3.

Property 2.2. $w_{lost} < k_3$

Proof. In the extended area of a bin belonging to $\mathcal{K}_3$, the total weight of elements that do not belong to a bin created by the greedy algorithm is strictly less than 1. Indeed, if it was not the case, another bin would have been created during phase 2. Therefore, $w_{\text{lost}} < k_3$. □

Furthermore, $|OPT_3| \leq w(OPT_3) \leq w(\mathcal{K}_3) + w(\mathcal{M}) + w_{\text{lost}}$ and since bins created by the greedy algorithm have weight at most 2, then $w(\mathcal{K}_3) < 2k_3$ and $w(\mathcal{M}) < 2m$. Hence,

$|OPT_3| < 2k_3+2m+k_3 < 3k_3+2m$ and finally $|OPT| < 3k_3+2m+2k_2 < 3(k+m)$,

which completes the proof of Lemma 2.1.

□

A thrifty refinement It is possible to further improve this approximation ratio to $\frac{2}{5}$ by imposing the following thrifty condition for bins created by the greedy algorithm. A bin of weight $1 + w$ is said to be *thrifty* if it only contains elements of weight more than w (it is thus not possible to remove any element and still have a valid bin). A thrifty bin has the following property:

Lemma 2.3. *If a thrifty bin B has $n(B)$ elements, then $w(B) < 1 + \frac{1}{n(B)-1}$*

Proof. Let $w(B) = 1 + w$. If B contains $n(B)$ elements of weights at least w, then $w(B) > n(B)w$ and thus $1 + w > n(B)w$. Therefore, $w < \frac{1}{n(B)-1}$. □

Lemma 2.4. *The greedy-and-thrifty algorithm gives a $\frac{2}{5}$-approximation algorithm of the* max_DCBC *problem with a resource augmentation of $2 + \frac{d}{d_{\max}}$.*

Proof. Lemma 2.3 can be used to bound more carefully $w(\mathcal{K}_3)$ in the proof of Lemma 2.1. Indeed, since every bin B of $\mathcal{K}_3$ has at least 3 elements, its weight $w(B)$ is at most $\frac{3}{2}$. Consequently, $w(\mathcal{K}_3) < \frac{3}{2}k_3$ and the final bound on $|OPT|$ becomes

$$|OPT| < \frac{5}{2}k_3 + 2m + 2k_2 < \frac{5}{2}(k + m),$$

which completes the proof of the Lemma 2.4. □

3 A distributed approximation algorithm

In this section, we describe how to organize nodes into an overlay network, so that the centralized algorithm with approximation ratio $\frac{1}{3}$ described in Section 2 can be turned into a distributed algorithm with low complexity. The resource augmentation factor of our algorithm is 4.

We assume our ambient metric space is $\mathbb{Q}^D$ for some (low) fixed dimension D, and we use the L_∞ norm to define distances. Thus, metric balls of radius r are products of intervals $[a_1 - r, a_1 + r] \times \cdots \times [a_D - r, a_D + r]$, and the whole space can be conveniently tiled by interior-disjoint balls.

3.1 The overlay

We use a *skip-list* as overlay. A skip-list [18, 15] is an ordered data structure based on a succession of linked lists with geometrically decreasing numbers of items. Skip-lists come into deterministic [15] and randomized [18] flavours. The deterministic versions have guaranteed properties whereas randomized skip-lists only offer high probability performance. The load-balancing effect introduced in the *skip-graph* variant [5] only works with the probabilistic version, so we use randomized skip-lists in the following.

The use of a skip-list requires an ordered set. Since the nodes lie in a D-dimensional space, we order them according to the Z-order [17] of their coordinates. Let us assume, without loss of generality, that all coordinates lie between 0 and 1; the Z-order is obtained by interleaving the binary expansions of the D coordinates of each point and by ordering the resulting strings lexicographically. We mostly use the following locality preserving property of the Z-order.

Property 3.1. *If* $\mathbf{x} = (x_1, \ldots, x_D)$ *is a point with coordinates of the form* $x_i = a_i/2^k$ *with integer* a_i*, then the product of intervals* $[x_1, x_1 + 2^{-k}) \times \cdots \times [x_D, x_D + 2^{-k})$ *is an interval in the* Z*-order.*

We call such a set a *level* k *ball* (see Figure 2). Note that a level k ball is indeed a ball in $\mathbf{R}^D$ with radius $2^{-(k+1)}$, and that each level k ball is the (interior-disjoint) union of 2^D level $k+1$ balls. As a consequence, $[0,1]^D$ is the interior-disjoint union of all 2^{kD} level k balls.

Let us also define a *level* k *skew ball* as a ball of radius $2^{-(k+1)}$ whose center is of the form $(a_i/2^{k+1})_{1 \leq i \leq D}$ for integer a_i, and a *level* k *doubly skew ball* as a ball of radius $2^{-(k+1)}$ whose center coordinates are of the form $a_i/2^{k+2}$ for integer k (see also Figure 2). Level k balls are exactly those level k skew balls for which all a_i are odd, and level k skew balls are exactly those level k doubly skew balls for which all a_i are even. Note that a level k skew ball is always the union of 2^D level $k+1$ balls, and thus, in the Z-order, the union of at most 2^D interior-disjoint intervals. A level k doubly skew ball is the union of 4^D level $k+2$ balls, and thus, the union of at most 4^D interior-disjoint invervals of the Z-order. Furthermore, taking all points at distance at most $2^{-(k+1)}$ of a given level k ball yields a level $k-1$ skew ball, and taking all points at distance at most $2^{-(k+1)}$ of a given level k skew ball yields a level $k-1$ doubly skew ball.

Let us now consider any $d_{\max} \leq 1$ of the form $d_{\max} = 2^{-k}$ for integer k. Then, for any subset S of $[0,1]^D$ of diameter at most $d_{\max}$, there exists a level $k-1$ skew ball which contains S.

3.2 The algorithm

In order to implement the 2-phases greedy algorithm presented in Section 2, we need to be able to greedily exhaust all bins with total weight at least 1 in all level $k-1$ skew balls, then in all level $k-2$ doubly skew balls, where $d_{max} = 2^{-k}$. The sketch of the algorithm for each host is made of three phases, where Phase 0 can be seen as a precomputing step in each level k ball.

Phase 0: Bins are created inside each level k ball.

Phase 1: Bins are created inside each level $k-1$ skew ball, using the remaining weights.

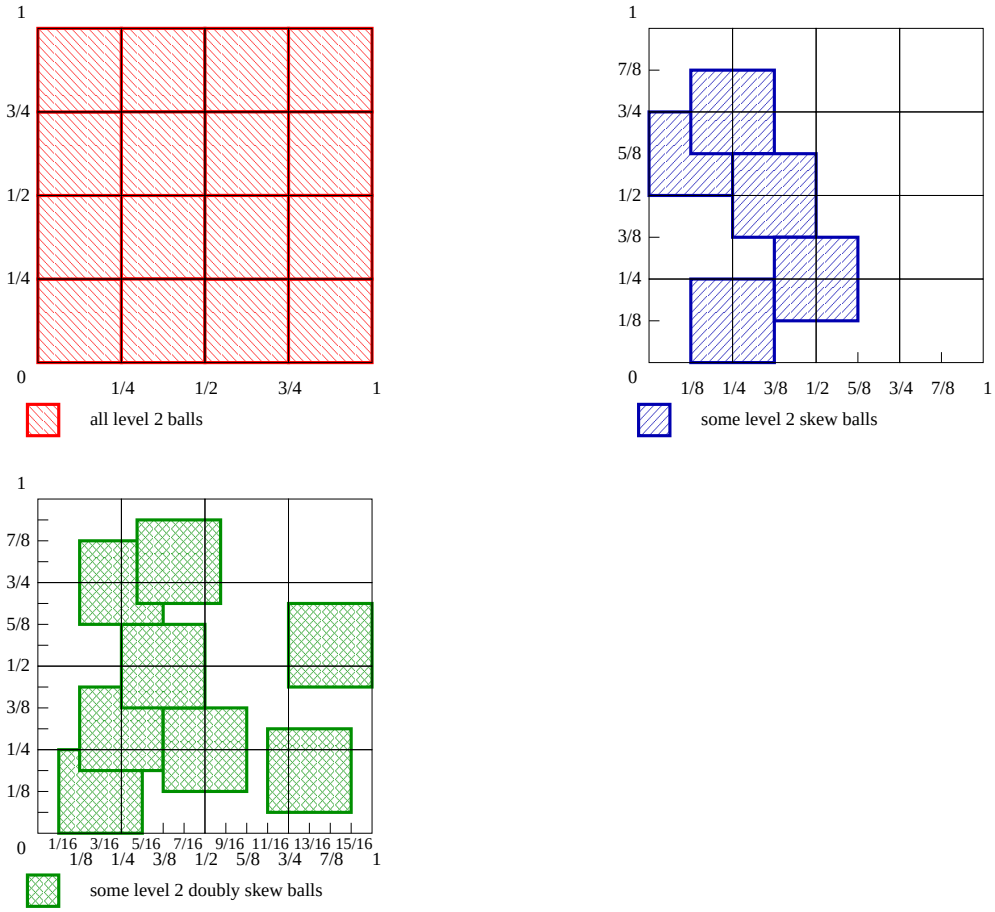


Figure 2: Example of balls and skew balls

Phase 2: Bins are created inside each level $k-2$ doubly skew ball, using the remaining weights.

Detail of Phase 0 Each host x computes the coordinates of the level k ball (thus of diameter d_{max}) it belongs to, and runs Algorithm 1 in this ball.

This algorithm uses the skip list to greedily cluster nodes in that ball; since we are operating inside the ball, the distance constraint is irrelevant. Each node x is assumed to have a weight $w(x)$, position $p(x)$ and height $h(x)$ in the skip list(*i.e.* the level of the highest list it appears in), and to know the corresponding information about each of its neighbors in the skip list. For this purpose, neighbors are predecessors $\text{pred}_i(x)$ and successors $\text{succ}_i(x)$ for each level $i \leq h(x)$.

Reasoning about the algorithm is easier in terms of the plane tree associated with the skip list. This tree has one node (x, i) for each level i that each node x appears in, and the ordered sons of (x, i) are all nodes $(y, i-1)$ such that $p(x) \leq y < p(\text{succ}_i(x))$. In Algorithm 1, each node goes up the tree and maintains the total weight W of ungrouped nodes in the subtree, together with a level ℓ under which all nodes in its subtree are already members of a bin. Most messages in the algorithm are actually sent from one node to one of its neighbours in the tree.

There are two main types of messages.

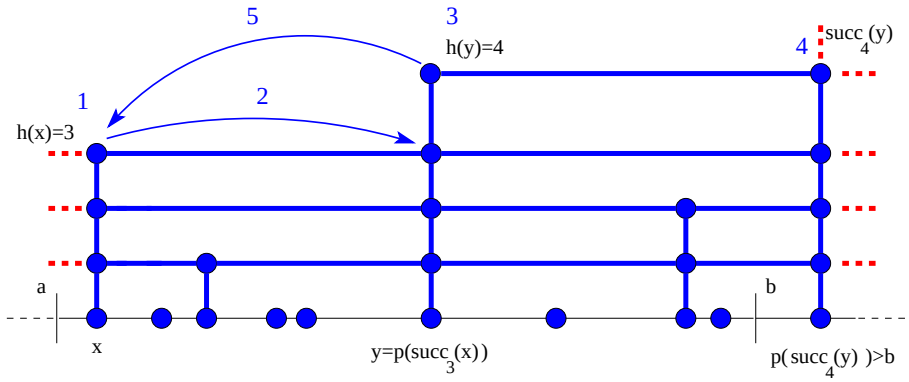
- WEIGHT messages indicate a level i, a candidate y, and a weight w. The meaning of such a message is that in the tree of height i rooted at the sender, there is a total ungrouped weight w. The node y is one of the ungrouped nodes, and will be used as a bin leader if this weight is used.
- BIN messages indicate a level i and a bin leader y. They are the counterpart of the weight messages, and mean that all ungrouped nodes in the tree of height i rooted at the receiver will be members of a common bin, with y as a leader.

During this phase, bins will be created inside each level k ball. At the end of this phase, the first host of the corresponding interval knows how much ungrouped weight is left in the ball. This host will be called the *coordinator* of this ball.

Detail of Phase 1 During this phase, all level $k-1$ skew balls will be scanned for remaining weight. Since these skew balls cover all sets of diameter at most $d_{\max}$, no potential bin of diameter $d_{\max}$ will remain at the end of this phase.

Remember that a level $k-1$ skew ball is the union of 2^D level k balls. The idea of this phase is to centralize all the weight information of a given $k-1$ skew ball at a single node. This node will then be able to decide which bins can be built. Scanning all the skew balls in parallel is not possible because they overlap; however we can partition them into 2^D sets $S_1, \ldots, S_{2^D}$ with the property that two balls in the same set do not overlap. Each level k ball is included in exactly 2^D level $k-1$ skew balls, one in each of these sets. Furthermore, we can arbitrarily assign to each skew ball a *coordinator* which will receive and process the corresponding information[2].

[2] A good example would be the leader of the lexicographically smallest level k ball it contains.



Execution:

1: beginning: $h(x) = 3$ and its successor at this level, y, is in $[a, b)$,

2: x sends (HEIGHT, x, b) to y, its successor at level 3,

3: y has a greater height, $h(y) = 4$,

4: the successor of y at level 4 is not in $[a, b)$,

5: thus y sends (HEIGHT-RESULT, $b, h(y)$) to x.

Figure 3: Execution of Algorithm 3.2

This phase will then consist in scanning all the 2^D sets in sequence. For a given set S_i, all level k ball leaders will report their remaining weight to the coordinator of the level $k-1$ ball of set S_i to which they belong. If the total weight is at least 1, this coordinator creates one or several bins (with weight strictly less than 2) and the corresponding BIN messages will be sent. Note that the remaining weight of a ball is never fractioned: it is either used in totality, or not at all.

The coordinator can be reached using the standard searching procedure in a skip-list, since its position is known given the skew ball it is responsible for. Performing all searches in parallel might prove costly for the highest level nodes in the skip-list. Therefore, we rely on the skip-graph variant [5], which provides a good load balancing in this case.

Detail of Phase 2 This phase is very similar to Phase 1. We scan all level $k-2$ doubly skew balls for remaining weight, thus ensuring that no more bins of diameter at most $3d_{\max}$ can be created. The idea is exactly the same, but this time the level $k-2$ doubly skew balls are partitioned into 4^D sets with the property that two balls in the same set do not overlap.

3.3 Complexity analysis

In the execution of Algorithm 1, each node sends a single WEIGHT message. Each BIN message from node x to node y is preceded by a WEIGHT message from y to x, so the total number of BIN messages is also n. The number of messages

Algorithm 1 Algorithm for greedy clustering in an interval $[a, b)$

For each node x:

1: **if** $p(\text{pred}_0(x)) < a$ **then**
2: compute maximum height $H(x)$ in interval $[a, b)$ using Algorithm 2
3: change its height in the skip list to $H(x)$
4: **end if**
5: $\ell \leftarrow -1$, $c \leftarrow x$, $W \leftarrow w(x)$
6: **for** i **from** 0 **to** $h(x)$ **do**
7: **if** $h(\text{succ}_i(x)) = i$ **and** $p(\text{succ}_i(x)) < b$ **then**
8: /* Remember $h(y)$ is the maximal height in which y appears in the skip list, therefore in general $h(\text{succ}_i(x)) \geq i$ */
9: receive message $(\textsc{Weight}, i, y, w)$ from $\text{succ}_i(x)$
10: $c \leftarrow y$, $W \leftarrow W + w$
11: **if** $W \geq 1$ **then**
12: send message $(\textsc{BinLeader}, W)$ to node y
13: call procedure $\textsc{CreateBin}(y, i)$
14: **end if**
15: **end if**
16: **end for**
17: send message $(\textsc{Weight}, i, c, W)$ to $\text{pred}_i(x)$
18: receive message $(\textsc{Bin}, y, i)$ and call procedure $\textsc{CreateBin}(y, i)$

Procedure $\textsc{CreateBin}(y, i)$:

1: **if** x does not yet belong to a bin **then**
2: send message $(\textsc{BinElement}, x, w(x))$ to y
3: **end if**
4: **for** j from ℓ to i **do**
5: **if** $h(\text{succ}_j(x)) = j$ **and** $p(\text{succ}_j(x)) < b$ **then**
6: send message $(\textsc{Bin}, y, j)$ to $\text{succ}_j(x)$
7: **end if**
8: **end for**
9: $\ell \leftarrow i + 1$, $W \leftarrow 0$

Upon reception of message $(\textsc{BinLeader}, W)$:

1: Wait for messages $(\textsc{BinElement}, x, w)$ with total w's equal to W; the corresponding x's are the bin elements

Algorithm 2 Computing maximum height in an interval $[a, b)$

Node x is assumed to be the first node in interval $[a, b)$, and to send itself a message $(\textsc{Height}, x, b)$.

For each node y, upon reception of a message $(\textsc{Height}, x, b)$:

1: **if** $p(\text{succ}_{h(y)}(y)) \geq b$ **then**
2: send message $(\textsc{Height-Result}, b, h(y))$ to node x
3: **else**
4: send message $(\textsc{Height}, x, b)$ to node $\text{succ}_{h(y)}(y)$
5: **end if**

used by each execution of Algorithm 2 (and by the resulting change in height) is bounded by the returned height, which is $O(\log n)$ in expectation and with high probability (whp for short). Thus, the overall number of message exchanges during the execution of Algorithm 1 on m disjoint intervals is $O(n + m \log n)$, both in expectation and whp.

In each interval, WEIGHT messages are relayed along a given level of the skip list until they reach a node with a higher level. Thus, the number of such relays for a given message follows a geometric distribution, with bounded expectation, and the longest relay sequence (formally, the highest k such that for some node x and level i, one has $\mathrm{succ}_i^k(x) = \mathrm{succ}_{i+1}(x)$) is, in expectation and whp, $\Theta(\log n)$. Together with the usual $O(\log n)$ bound on the number of levels in a skip-list, this implies the following result.

Theorem 3.2. *Algorithm 1, run in parallel for any number of m disjoint intervals, uses $O(n + m \log n)$ messages and, in a synchronous execution model where each message takes unit time, $O(\log^2 n)$ rounds, both in expectation and with high probability* [3].

For later phases, very few messages are sent except for BIN messages, which are already accounted for in our previous analysis, and messages for searching for ball leaders in the skip-list. The expected complexity (and here, number of messages) for a search in a skip-list of size n is $O(\log n)$, and we are performing $O(m)$ searches, for a total of $O(m \log n)$ messages; and the longest search is $O(\log^2 n)$. Thus, Theorem 3.2 also holds for the whole computation.

4 Conclusion

We have proposed a fully distributed algorithm that builds small clusters, such that in each cluster, the aggregated capacity is larger than a threshold and the maximal distance between two nodes belonging to the same cluster is smaller than a given threshold $d_{\max}$. The approximation ratio for this algorithm is $\frac{1}{3}$, if we compare the number of clusters (with maximal diameter 3 $d_{\max}$) created by the distributed algorithm to the optimal number of clusters with diameter $d_{\max}$. Moreover, the complexity of the distributed algorithm is kept relatively low when the metric space is $\mathbb{Q}^D$ (for a small value of D) and the L_∞ norm is used to define distances. This algorithm takes $O((4^D) \log^2 n)$ rounds and $O((4^D) n \log n)$ messages both in expectation and with high probability, where n is the total number of hosts. Last, the distributed algorithm, although rather sophisticated, can be implemented in practice since it only relies on classical distributed data structures such as skip graphs [5].

In future works, we plan to adapt the algorithm to the case where several resource requirements must be satisfied simultaneously (for instance, a task may require both a large aggregated memory and a large disk storage capacity). Another interesting work is to compare the performance of the distributed algorithm we propose with the gossip-based approach. Gossip-based algorithm complexities are usually very difficult to establish, but these algorithms have been proved to be very efficient to exploit locality [14, 19]. Finally, we need to

[3] Note that the $O(\log^2 n)$ time bound could very likely be brought down to $O(\log n)$ by a more careful analysis (nodes with large height are rare, as are long sequences of consecutive nodes on a level i list with no nodes of higher level among them).

adapt the algorithm to the case where the metric space is not $\mathbb{Q}^D$. Indeed, if network coordinates systems based on landmarks [16] used $\mathbb{Q}^D$ (for values of D of order 10) as underlying metric space, more recent coordinate systems, such as Vivladi [8] rely on much more sophisticated metric spaces (but based on 3 coordinates only).

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] O. Beaumont, N. Bonichon, P. Duchon, and H. Larcheveque. Distributed approximation algorithm for resource clustering. In LNCS Series Springer, editor, *Proceedings of SIROCCO'08*, 2008.

[8] 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.

[9] 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.

[10] János Csirik and Gerhard J. Woeginger. Resource augmentation for online bounded space bin packing. *J. Algorithms*, 44(2):308–320, 2002.

[11] 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.

[12] Leah Epstein and Rob van Stee. Online bin packing with resource augmentation. In Giuseppe Persiano and Roberto Solis-Oba, editors, *WAOA*, volume 3351 of *Lecture Notes in Computer Science*, pages 23–35. Springer, 2004.

[13] M. Franceschetti, M. Cook, and J. Bruck. A geometric theorem for approximate disk covering algorithms, 2001.

[14] A.J. Ganesh, A.M. Kermarrec, and L. Massoulié. Peer-to-Peer Membership Management for Gossip-Based Protocols. 2003.

[15] 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.

[16] T.S.E. Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In IEEE, editor, *Proceedings of INFOCOM'02*, pages 170–179, 2002.

[17] Jack Orenstein. A comparison of spatial query processing techniques for native and parameter spaces. *SIGMOD Rec.*, 19(2):343–352, 1990.

[18] W. Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6), 1990.

[19] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex (France)

Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399