# Allocation of Publisher/Subscriber Data Links on a Set of Virtual Machines

Thomas Lambert and Rizos Sakellariou
School of Computer Science, The University of Manchester, Manchester, UK
thomas.lambert@manchester.ac.uk, rizos@manchester.ac.uk

***Abstract*—There is an increasing interest in applications where sensors or other devices (acting as publishers) may generate data that are processed and analyzed by specialized software components (acting as subscribers to this data) that extract useful information in a variety of scientific or industrial settings. As a result of the large volumes of data that are often generated, Cloud infrastructures may be used to handle the data links between publishers and subscribers. Assuming that a certain number of virtual machines of some given bandwidth have been booked for this purpose, the problem that this paper considers is how to allocate data links to the virtual machines so that the amount of data received by subscribers is maximized. An Integer Linear Programming formulation of the problem and two heuristics are presented, which are evaluated in a range of experiments.**

## I. Introduction

The publish/subscribe model (shortened as pub/sub) has been a popular paradigm for many applications exchanging messages. In recent years, the advent of the Internet-of-Things (IoT) and other data streaming applications where sensors may continuously provide data that have to be processed and analyzed has given prominence to the data-centric aspects of the pub/sub model [1]. In brief, the pub/sub model should be viewed as a model that allows simple modelling of the interaction(s) between a set of data providers (publishers) and a set of (possibly only partially) interested data sinks (subscribers) [2].

Traditionally, the pub/sub model has been widely used to accomplish various messaging tasks. The move towards data-centric applications and requirements in conjunction with the development in recent years of Cloud Computing offer enterprises the possibility to run pub/sub-based applications and services on external clusters. Some Infrastructure-as-a-Service (IaaS) providers already propose generic pub/sub engines (e.g., PubNub or Azure Service Bus). In other research, the Cloud is used to build integrated pub/sub middleware solutions [1]. To support this, it is safe to assume that some virtual machines (VMs) are provided to the clients to run their application and satisfy their pub/sub requirements; obviously, the number of VMs depends on what they paid.

The above developments, most notably the data-centric nature of pub/sub applications and the availability of Cloud platforms for (some components or whole) application execution, give rise to the following interesting problem. *Assuming that a fixed number of VMs, each VM with a certain bandwidth capability, has been booked to transmit data from publishers to subscribers interested in this data, how do we allocate transmissions in a way that we maximize satisfaction of subscribers?* The satisfaction of subscribers may be expressed in different forms but in this paper we equate satisfaction with the amount of data received. As a result, our question becomes a problem of allocating the pub/sub data links on a fixed set of VMs, in a way that the amount of data received by subscribers is maximized. In the rest of the paper, we denote this problem by the shorthand MaxDataRcvd. The amount of data generated by providers may exceed what can be received by subscribers or the capacity of the available VMs: such cases make the problem more challenging to solve.

To the best of our knowledge, this problem has not been investigated as such in the literature. The closest work to our paper is by Setty et al. [3], who considered strategy allocations to optimize cost by deploying the smallest possible number of VMs in similar pub/sub settings. The MaxDataRcvd problem we consider can be viewed as a dual problem. In other words, given publishers (that publish at a certain rate), subscribers (that receive data from certain publishers) and machines (their number and capabilities are fixed), how can we allocate the links so that we maximize the amount of data received by subscribers?

In view of the above, this paper makes the following contributions: (i) it provides a formulation of this data link allocation problem and expresses it in a way that can be solved through Integer Linear Programming (ILP); (ii) as ILP-based solutions may be costly, it proposes two new heuristics to solve the problem; and, (iii) it carries out an experimental evaluation that assesses the behaviour of an ILP-based solution and demonstrates that the heuristics can achieve good performance (compared to the optimal ILP-based solution) but much faster, with reasonable computation time and scalability properties.

The rest of the paper is structured as follows. Section II briefly overviews related work. In Section III, a formal model of the problem and its Integer Linear Programming formulation are presented. In Section IV, two heuristics are proposed to solve the problem. An experimental evaluation is carried out in Section V. Finally Section VI concludes the paper and gives some perspective of this work.

## II. Related Work

The paper touches upon a number of areas where there is extensive work in the literature. Middleware solutions for the IoT that are based on the pub/sub model have received some

attention [4], [5], [6]. A lot of work exists in relation to streaming applications that make use of some pub/sub system [7], [8]. The problem we are considering goes beyond this research, as it takes it into account in order to focus on an allocation problem specified (and affected) by pub/sub properties. In some way, MAXDATARCVD resembles the Multiple Knapsack Problem (MKP). In MKP, we have a set of items, each having a profit and a weight, and a set of bins, each having a capacity. The goal is to allocate the items to the bins trying to respect the capacity constraint while maximizing the profit. MKP is a generalisation of the Knapsack Problem, where there is only one bin, and it is an NP-complete problem [9]. The differences between MKP and MAXDATARCVD are discussed in Section III. There are polynomial-time approximation schemes for some instances of MKP and its variants [10]. What is interesting with respect to the problem considered in the paper is that heuristic approaches have also been considered for MKP [11], [12] and some branch-and-bound based solutions exist that focus on a small number of bins [13], [14]. We are going to build upon some of this work in Section IV.

## III. MODEL AND ILP FORMULATION

We consider a bipartite graph $G = (P, S, E)$ with $P$ being the set of *publishers* and $S$ being the set of *subscribers*. Each publisher $p_i$ has a rate $r_i$, which is the rate with which it generates data; we refer to this rate as *input rate*. We also consider a set of (virtual) machines $M$. Each machine $m_k$ has a *bandwidth capacity* $b_k$. Subscribers are interested in data provided by only certain publishers; an edge (i.e., a data link) connects every such pair of publisher/subscriber. A publisher may generate data for many subscribers and a subscriber may be interested in data provided by different publishers. The set of all edges is $E$.

As stated in the introduction, our goal is to allocate the different edges to the different machines in a way that we do not exceed the bandwidth capacity of each machine and we maximize the amount of data received by publishers. We are, therefore, looking for an *allocation* $\sigma$ as a function from $E' \subseteq E$ to $M$, with $E' \subseteq E$. In addition, as an edge allocated to a (virtual) machine implies receiving and sending data, for all edges of each machine the sum of incoming data (the input rate of publisher for each edge) and outgoing data (based again on input rate of the publisher of this edge, as we assume the machines will not do any filtering of the data) must be less than or equal to bandwidth capacity, leading to:

$$\forall m_k \in M, \sum_{p_i \in P_\sigma(m_k)} r_i + \sum_{p_i s_j \in E_\sigma(m_k)} r_i \leq b_k,$$

where $P_\sigma(m_k)$ denotes the set of publishers $p_i$ such that there exists $s_j \in S$ such that $p_i s_j \in E'$ and $E_\sigma(m_k)$ is the set of edges allocated to $m_k$.

An example is given in Figure 1, which represents an allocation of edges onto machines. Publishers are shown on the left-hand side, each with an input rate $r_i$. Subscribers are shown on the right-hand side. Edges denoted by a solid line are allocated ($\in E'$) to a machine (represented by the rectangles in the middle of the figure), edges with a dashed line are not (for example $p_6 s_6 \notin E'$). The constraints in this schedule are:

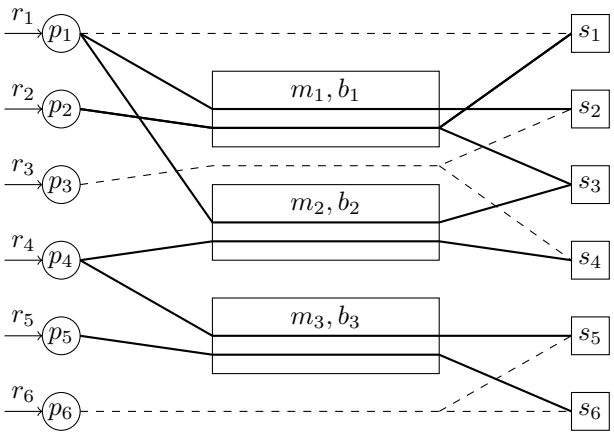


Figure 1. An illustration of a schedule for MAXDATARCVD.

- $(r_1 + r_2) + (r_1 + 2r_2) \leq b_1$.
- $(r_1 + r_4) + (r_1 + r_4) \leq b_2$.
- $(r_4 + r_5) + (r_4 + r_5) \leq b_3$.

Note that the bandwidth usage of a schedule depends, for each publisher, on the number of machines on which its edges are scheduled. If all the allocated edges of a publisher are on the same machine, the input rate of this publisher contributes to bandwidth usage only once. If the edges are allocated on several machines, the input rate contributes to the bandwidth usage of each machine separately, thus, increasing the impact of this publisher on the overall bandwidth consumption. For example, in Figure 1, publisher $p_1$, whose data is sent to $m_1$ and $m_2$, contributes to the bandwidth usage of these two machines, whereas $p_2$, for instance, contributes to bandwidth usage of only $m_1$. The fact that the bandwidth consumption of a publisher depends on how edges are scheduled is where the difference between MAXDATARCVD and MKP stands. In our problem, the "items" to allocate are the edges, their "profits" are the input rates of the publishers, but their "weights" are not constant. Depending on the placement of other "items" (edges with the same publisher) the "weights" may change.

Having defined the constraints, we can now introduce the formal definition of our problem, MAXDATARCVD, which simply becomes the search of such a scheduling that maximizes the amount of data received, *i.e.* the sum of the input rates of allocated edges.

**Problem 1** (MAXDATARCVD)**.** *Given $G = (P, S, E)$ a bipartite graph with a given incoming rate $r_i$, $M$ a set of virtual machines with, for each virtual machine $m_k$, bandwidth capacity $b_k$, return $E' \in E$ and an allocation $\sigma$ which maximize*

$$\sum_{p_i s_j \in E'} r_i.$$

The MAXDATARCVD problem can be converted into an Integer Linear Programming (ILP) problem with linear size

$O((|E|+|P|)|M|)$. Keeping the previous definition, we define the following decision variables:

- For each edge $p_i s_j \in E$ and each machine $m_k$, we have $x_{i,j,k} \in \{0,1\}$; in the final schedule $\sigma(p_i s_j) = m_k$ if and only if $x_{i,j,k} = 1$.
- For each publisher $p_i$ and each machine $m_k$, we have $y_{i,k} \in \{0,1\}$; $y_{i,k} = 1$ if and only if there is an edge $p_i s_j$ such that $\sigma(p_i s_j) = m_k$.

The expression to maximize is therefore:

$$\sum_{\substack{p_i s_j \in E \\ m_k \in M}} x_{i,j,k} r_i$$

under the constraints:

- $\forall p_i s_j \in E, \sum_{k \in [1,|M|]} x_{i,j,k} \leq 1.$
- $\forall p_i \in P, \sum_{j,\ p_i s_j \in E, k \in [1,|M|]} x_{i,j,k} \leq y_{i,k}.$
- $\forall m_k \in M,\ r_i \left( \sum_{i \in [1,|P|]} y_{i,k} + \sum_{p_i s_j \in E} x_{i,j,k} \right) \leq b_k.$

## IV. Heuristics

As ILP solutions may be too costly, this section introduces two heuristics to solve the problem (an evaluation of an ILP solution is given in Section V). In the following, given a publisher $p_i$ (respectively, a subscriber $s_j$), we denote by $S_i$ the set of subscribers that have an edge in $E$ with $p_i$ (respectively, $P_j$ is the set of publishers that have an edge in $E$ with $s_j$).

**Algorithm 1:** GS

**Input:** A bipartite graph $G = (P, S, E)$, incoming rates $\{r_1, \ldots, r_{|P|}\}$, a set of machines $M$ and $f : E \to \mathbb{R}$
**Output:** $E' \in E$ and an allocation $\sigma$

```
Sort P by decreasing number of edges ;
Sort M by decreasing bandwidth capacity ;
E', σ = ∅ ;
while E ≠ ∅ and M ≠ ∅ do
    p_i ← Pop(P) ;
    m_k ← Pop(M) ;
    if b_k ≥ 2 × r_i then
        b_k ← b_k − r_i ;
        while b_k ≥ r_i and S_i ≠ ∅ do
            s_j ← Pop(S_i) ;
            Add p_i s_j to E' ; σ(p_i s_j) = m_k ;
            b_k = b_k − r_i ;
        if b_k > 0 then
            Push(m_k, M) ;
        if |S_i| > 0 then
            Push(p_i, P) ;
return E', σ ;
```

The first heuristic, GS (GreedyScheduler, see Algorithm 1) relies on a greedy allocation of edges. The principle is to sort the publishers by number of edges in descending order, take the first one (that is the publisher with most edges) and allocate as many edges as possible onto the machine with the largest bandwidth capacity. After that, if not all edges of this publisher have been scheduled, this publisher is pushed back to $P$ after updating its number of edges. Similarly, if the bandwidth capacity of the machine is not zero after allocating some work onto it, this machine is also pushed back to $M$. At each step of the while loop, two operations are performed: the allocation of at least one edge (an edge is allocated as soon at it is visited and thus the overall cost of this operation in the overall execution is $O(|E|)$) and the insertion of a machine or a publisher into a sorted list (respectively $O(\log |M|)$ and $O(\log |P|)$). Thus, as there are at most $|E|$ steps, the overall complexity is $O(|E|(\log |M| + \log |P|))$ if $|P|, |M| \leq |E|$. Otherwise, the cost of the initial sorting of publishers and machines has to be taken into account.

The second heuristic, MKPBS (MKP-Based-Scheduler, see Algorithm 2), is based on the proximity between MaxDataR-cvd and MKP. The principle is to use an MKP solver to schedule the publishers onto different machines. We do this in two main phases. First we consider a publisher and all its edges as one entity (we schedule all of them on one machine or we schedule none of them). In this case, the items are the publishers (and its edges); then, for each publisher, the profit is given by its number of edges multiplied by its input rate and the weight is its input rate multiplied by its number of subscribers plus one. After this first phase we have a first schedule (possibly empty). During the second phase, we reuse an MKP solver, but this time we only allocate publishers and then we deal with edges one by one. We consider publishers as items with a profit equal to their input rates multiplied by their number of edges (best-case scenario if all are scheduled) and a weight equal to twice their input rate (the initial cost to send data and the cost for at least one scheduled edge). This gives us a repartition of publishers. We then use this repartition to schedule edges, using Knapsack sub-problems. The instances are built as follows: items are edges (profit and weight are equal to the input rate) and the capacity is the bandwidth capacity of the machine minus the input rates of the allocated publishers. This Knapsack instance gives us a schedule for some of the edges. Note that in some cases, there are allocated publishers with no scheduled edges. In such a case, we remove the publisher from the allocation (and then increase the bandwidth) and try to greedily schedule edges from the other allocated publishers. We then repeat the second phase until there is no publisher allocated by the MKP solver.

In the following, we denote by MKPSolver the MKP solver and by KPSolver the Knapsack solver we use. For the implementation we use a branch-and-bound exact solver as Knapsack solver (at each call, the number of items is rather small). The algorithm is called MulKnap (this is in fact a more general solver designed for MKP) and has been proposed by David Pisinger [13] (the code can be found on his webpage). For the MKP solver we rely on a greedy heuristic, GreedyMKP (MulKnap has a long computation time for our

**Algorithm 2:** MKPBS

**Input:** A bipartite graph $G = (P, S, E)$, incoming rates $\{r_1, \ldots, r_{|P|}\}$, a set of machine $M$ and $f : E \to \mathbb{R}$
**Output:** $E' \in E$ and an allocation $\sigma$

$E', \sigma = \emptyset$ ;
**foreach** $p_i \in P$ **do**
  $a_i.weight = (|S_i| + 1)r_i$ ; $a_i.profit = |S_i|r_i$ ;
$I_1, \ldots, I_{|M|} \leftarrow$
 $MKPSolver(\{a_1, \ldots, a_{|P|}\}, \{b_1, \ldots, b_{|M|}\})$;
**foreach** $m_k \in M$ **do**
  **foreach** $p_i \in I_k$ **do**
    $b_k \leftarrow b_k - (|S_i| + 1)r_i$ ;
    Add $S_i$ to $E'$ ; $\sigma(S_i) = m_k$;
    Remove $p_i$ from $P$ ;
**while** $P \neq \emptyset$ *and* $M \neq \emptyset$ **do**
  **foreach** $p_i \in P$ **do**
    $a_i.weight = 2r_i$ ; $a_i.profit = |S_i|r_i$ ;
  $I_1, \ldots, I_{|M|} \leftarrow$
   $MKPSolver(\{a_1, \ldots, a_{|P|}\}, \{b_1, \ldots, b_{|M|}\})$ ;
  **if** $\forall m_k,\ I_k = \emptyset$ **then**
    $P = \emptyset$ ;
  **else**
    **foreach** $m_k \in M$ **do**
      **foreach** $a_i \in I_k$ **do**
        **foreach** $s_j \in S_i$ **do**
          $a'_j.weight = r_i$ ; $a'_j.profit = r_i$ ;
      $I \leftarrow$
       $KPSolver(\{\ldots, a'_j, \ldots\}, b_k - \sum_{a_i \in I_k} r_i)$ ;
      **foreach** $a'_j \in I$ **do**
        Add $p_i s_j$ to $E'$ ; $\sigma(p_i s_j) = m_k$;
        Remove $s_j$ from $S_i$ ;
        $b_k = b_k - r_i$ ;
        **if** $p_i \notin P_\sigma(m_k)$ **then**
          Add $p_i$ to $P_\sigma(m_k)$ ; $b_k = b_k - r_i$ ;
      **foreach** $p_i \in P_\sigma(m_k)$ **do**
        **if** $S_i \neq \emptyset$ *and* $r_i \leq b_k$ **then**
          $s_j \leftarrow Pop(S_i)$ ;
          Add $p_i s_j$ to $E'$ ; $\sigma(p_i s_j) = m_k$ ;
          $b_k = b_k - r_i$ ;
    **foreach** $p_i \in P$ **do**
      **if** $S_i = \emptyset$ **then**
        Remove $p_i$ from $P$ ;
    **foreach** $m_k \in M$ **do**
      **if** $b_k = 0$ **then**
        Remove $b_k$ from $M$ ;
**return** $E', \sigma$ ;

**Algorithm 3:** MKPSolver

**Input:** Items $A = \{a_1, \ldots, a_n\}$, capacity $C = \{c_1, \ldots, c_l\}$
**Output:** For each $k \in [1, l]$ a set $I_k \in [1, n]$ of allocated items

Sort $A$ by decreasing $a_i.profit/a_i.weight$ ;
Sort $C$ by decreasing capacity ;
**foreach** $c_k \in C$ **do**
  $I_k = \emptyset$ ;
  **foreach** $a_i \in A$ **do**
    **if** $a_i.weight \leq c_k$ **then**
      $I_k \leftarrow I_k \cup \{a_i\}$ ;
  $A \leftarrow A \setminus I_k$ ;
**return** $I_1, \ldots, I_k$ ;

instances), proposed by Martello and Toth [11], that has a good behaviour in practice. Basically, this heuristic sorts the items by decreasing rentability ($profit/weight$, that is, in our case, equivalent to sorting by decreasing number of edges) and the bins by decreasing capacity. Then, one bin after another, one item after another, GreedyMKP allocates an item if its weight is smaller than the current capacity of the concerned bin (see Algorithm 3).

It is not straightfoward to evaluate the complexity of MKPBS. Using GreedyMKP, the cost of each MKP sub-instance is simply $|M||P|$. The number of times that we call GreedyMKP is at most $O(|E|)$ times, as at each call at least one edge is allocated. Meanwhile, the cost of other operations is strongly correlated to the cost of KPSolver. The problem can be solved in pseudo-polynomial time $O(nc)$ where $n$ is the number of items and $c$ the capacity. Thus, the cost of solving all Knapsack instances during one step can be bound by $O((\max b_k)|E|)$ and the overall worst-case complexity of MKPBS is $O(|E|((\max b_k)|E| + |M||P|))$.

## V. Experimental Evaluation

The objectives of the experimental evaluation are to: (i) investigate under what conditions ILP may give answers within some reasonable computation time; (ii) evaluate the performance of GS and MKPBS in comparison to the optimal ILP result and to an upper bound we describe below; and (iii) assess computation time and scalability of both GS and MKPBS.

We use a small instance of a synthetic dataset in order to avoid long execution times for the ILP solution as we wish to compare ILP results with the results provided by our heuristics. The number of subscribers, $|S|$, is set to 100 and two different values for the number of publishers, $|P|$, are used: 100 and 1000. We assume all subscribers get data from exactly 5 publishers (the number of edges is therefore 500, independently of the number of publishers). We also suppose identical machines (with a varying number of machines, $|M|$), all with the same bandwidth (bandwidth units are assumed to be the same with

| | $\forall m_k, b_k = 200$ | $\forall m_k, b_k = 500$ |
|---|---|---|
| Homogeneous | 50 times for $\|M\| = 5$<br>7 times for $\|M\| = 10$<br>no other values tried | 50 times for every $\|M\|$ |
| LowVariance | 0 times for $\|M\| = 5$<br>no other values tried | 15 times for $\|M\| = 5$<br>39 times for $\|M\| = 10$<br>no other values tried |
| HighVariance | 49 times for $\|M\| = 5$<br>11 times for $\|M\| = 10$<br>no other values tried | 48 times for $\|M\| = 5$<br>46 times for $\|M\| = 10$<br>0 times otherwise |

Table I

NUMBER OF TIMES THE ILP PROGRAM FINDS A SOLUTION WITHIN THE TIME LIMIT FOR DIFFERENT VALUES OF BANDWIDTH AND DISTRIBUTION OF INPUT RATES WHEN $|P| = |S| = 100$.

| | $\forall m_k, b_k = 200$ | $\forall m_k, b_k = 500$ |
|---|---|---|
| Homogeneous | 45 times for $\|M\| = 10$<br>37 times for $\|M\| = 15$<br>50 times otherwise | 50 times for every $\|M\|$ |
| LowVariance | 29 times for $\|M\| = 5$<br>no other values tried | 34 times for $\|M\| = 5$<br>49 times for $\|M\| = 15$<br>50 times otherwise |
| HighVariance | 49 times for $\|M\| = 5$<br>18 times for $\|M\| = 10$<br>no other values tried | 49 times for $\|M\| = 5$<br>50 times otherwise |

Table II

NUMBER OF TIMES THE ILP PROGRAM FINDS A SOLUTION WITHIN THE TIME LIMIT FOR DIFFERENT VALUES OF BANDWIDTH AND DISTRIBUTION OF INPUT RATES WHEN $|P| = 1000$ AND $|S| = 100$.

input rate). We consider three distributions for the input rate of publishers: equal to 10 (Homogeneous), uniformly distributed from 8 to 19 (LowVariance) and uniformly distributed from 1 to 19 (HighVariance). For each combination, we run the experiment 50 times (each time generating a new set of edges $E$) and we take the average value.

The performance of the algorithms is expressed in terms of the average percentage of data received, which is the amount of data received by the subscribers divided by the sum of the input rates of all edges (including non-allocated edges). To have an upper bound for comparison, we take the minimum value of the sum of the bandwidths of all machines and the sum of the input rates of all edges.

For the solution of the ILP problem, the program we use is the CPLEX optimizer provided by IBM [15]. CPLEX also exploits parallelism that we use to run the computation on a 24 processor (MIRIEL node, composed of two Dodeca-core Haswell Intel Xeon E5-2680 v3 at 2.50 GHz) of PlaFRIM2 [16].

### A. Analysis of the ILP Behaviour

For the first set of experiments, we try ILP using our synthetic dataset and assuming machines of bandwidth 200 and 500. We vary the number of machines, $|M|$, from 5 to 50 in steps of 5. In order to avoid long computation times, we set a time limit of one hour for the ILP program to find a solution.

Tables I and II show the number of times that the ILP program finds a solution within the time limit. As expected, ILP cannot be used to solve MAXDATARCVD efficiently, even for small or homogeneous cases. Even with a small number of machines the problem presents sufficient combinatorial complexity that routinely leads to an explosion of the computation time. When the number of machines increases and there is more bandwidth available it appears that the computation time may become reasonable; this is easily explained as in this case it is easier and more likely to find an optimal solution. We observe this behaviour when the value of bandwidth per machine is 500.

### B. Performance of GS and MKPBS

In the following, we compare GS and MKPBS with the optimal results we have from ILP. As already mentioned, performance is expressed in terms of the percentage of data received, which is the amount of data received by the subscribers divided by the sum of the input rates of all edges (including non-allocated edges).

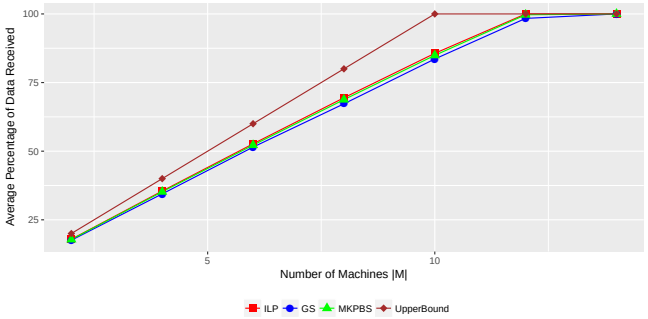


Figure 2. Average percentage of received data for ILP, GS and MKPBS. All machines have bandwidth equal to 500 and $|P| = |S| = 100$.

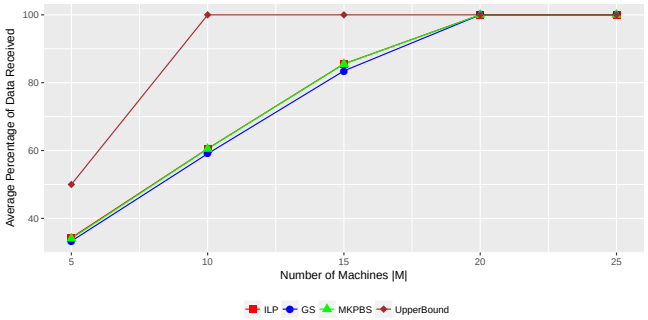


Figure 3. Average percentage of received data for ILP, GS and MKPBS. All machines have bandwidth equal to 500, $|P| = 1000$ and $|S| = 100$.

We focus first on the Homogeneous distribution with bandwidth equal to 500 as it is the configuration where ILP is most successful for both $|P| = 100$ and $|P| = 1000$. The results are presented in Figures 2 and 3. Both GS and MKPBS perform very close to the optimal solution computed by ILP. Most specifically, MKPBS is always almost optimal in the case where $|P| = 1000, |S| = 100$ (Figure 3). This last result can be explained by the fact that with few subscribers per publisher, the instance become closer to an MKP instance: if a publisher has only one or two edges, it is more likely to

have all its edges scheduled on the same machine and get an optimal solution. However, the heuristics perform very well with a more dense distribution of subscribers too (Figure 2). To give a clearer indication of the results from the figures, some points are extracted and shown in Tables III and IV.

| | $\|M\| = 5$ | $\|M\| = 10$ | $\|M\| = 15$ |
|---|---|---|---|
| ILP | 44.11% | 85.58% | 100% |
| GS | 42.99% | 83.69% | 100% |
| MKPBS | 43.78% | 84.95% | 100% |

Table III
AVERAGE PERCENTAGE OF DATA RECEIVED FOR SPECIFIC VALUES OF $|M|$. ALL MACHINES HAVE BANDWIDTH EQUAL TO 500 AND $|P| = |S| = 100$ (SEE FIGURE 2)

| | $\|M\| = 5$ | $\|M\| = 10$ | $\|M\| = 15$ | $\|M\| = 20$ |
|---|---|---|---|---|
| ILP | 34.30% | 60.56% | 85.56% | 100% |
| GS | 33.29% | 59.07% | 83.49% | 100% |
| MKPBS | 34.04% | 60.50% | 85.50% | 100% |

Table IV
AVERAGE PERCENTAGE OF DATA RECEIVED FOR SPECIFIC VALUES OF $|M|$. ALL MACHINES HAVE BANDWIDTH EQUAL TO 500, $|P| = 1000$ AND $|S| = 100$ (SEE FIGURE 3)

Besides comparison between heuristics and ILP, we observe that more publishers also imply more bandwidth consumption. This is why for the same number of machines, in the case of $|P| = 1000$ a smaller percentage of received data is obtained. Note also that the performance of both MKPBS and GS does not appear to be linked to the variance of input rates, as can be seen in Figure 4 (only the case $|P| = 100$ is shown to be compared against results for Homogeneous in Figure 2).

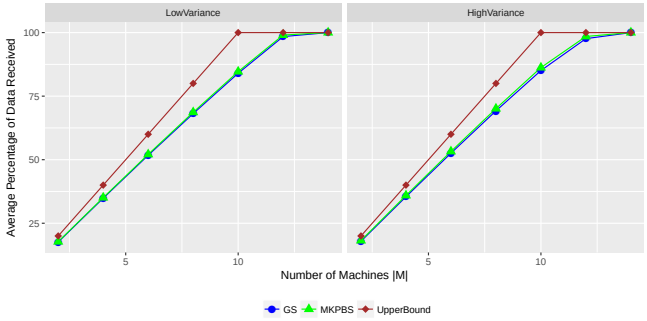


Figure 4. Average percentage of received data, depending on the distribution of input rates for GS and MKPBS. All machines have bandwidth equal to 500 and $|P| = |S| = 100$.

Finally, we also examine the influence of the value of bandwidth. Figure 5 shows the results where $|P| = 1000$, $|S| = 100$ and the bandwidth of each machine is 200. Compared to the results in Figure 3, it can be seen that the average percentage of data received for 10 or 15 machines is about 2.5 times less, a value proportional to the difference in bandwidth. The same remark applies to the case where $|P| = 100$ (Figure 6). It is also interesting to note that in both Figures 5 and 6, the performance difference between MKPBS and GS is more pronounced.

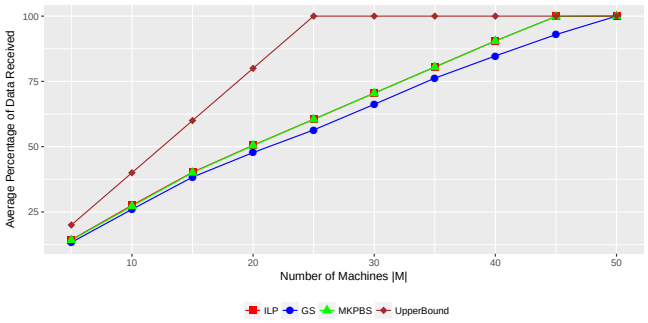


Figure 5. Average percentage of received data for ILP, GS and MKPBS. All machines have bandwidth equal to 200, $|P| = 1000$ and $|S| = 100$.

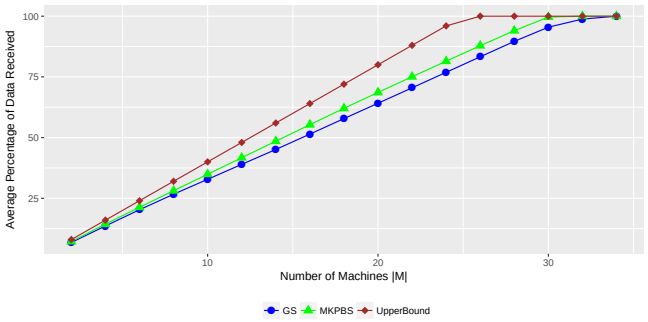


Figure 6. Average percentage of received data for GS and MKPBS. All machines have bandwidth equal to 200, $|P| = 100$ and $|S| = 100$.

### C. Computation Time

The following studies the computation time of both GS and MKPBS. All results come from a C implementation running on a Intel® Core™ i7-4600M processor. Results for all input rate distributions and with a bandwidth per machine equal to 500 are shown in Figures 7 and 8.

In principle, both heuristics are fast: below 1ms on average. However, GS appears to be faster and more consistent in terms of performance. The peaks in the performance of MKPBS can be attributed to the number of calls to KPSolver and MKPSolver. When there are enough machines, the initial call to MKPSolver has a strong chance to find a good schedule;

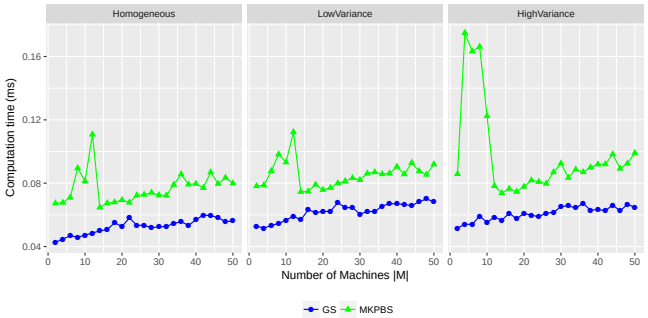


Figure 7. Average computation time in ms for GS and MKPBS. All machines have bandwidth equal to 500 and $|P| = |S| = 100$.

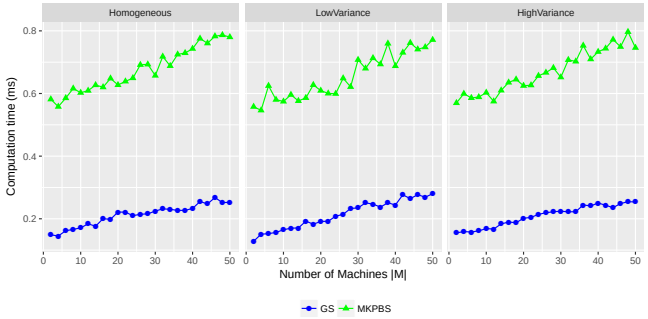


Figure 8. Average computation time in ms for GS and MKPBS. All machines have bandwidth equal to 500, $|P| = 1000$ and $|S| = 100$.

several calls may be necessary when the amount of available bandwidth (correlated with the number of machines) is limited (the peak is before reaching the value of $|M|$ for which MKPBS is able to schedule all edges). After this instability phase, the computation time slightly increases as the number of machines increases (consistent with the complexity of MKPSolver); a similar behaviour for MKPBS can be noted (consistent to the $\log |M|$ factor in its worst-case complexity). Meanwhile, the number of publishers has the main impact on computation time, which differs by about an order of magnitude between Figures 7 and 8.

Finally, it appears that a reduction of the bandwidth of each machine increases the magnitude of instability of MKPBS but it has no effect on GS as can be seen in Figure 9.

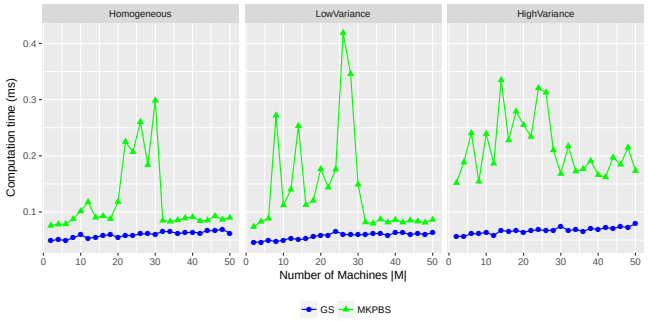


Figure 9. Average computation time in ms for GS and MKPBS. All machines have bandwidth equal to 200 and $|P| = |S| = 100$.

### D. Scalability

In this section, we study the scalability of GS and MKPBS. To do so, we multiply the amount of bandwidth and numbers of publishers, subscribers, edges per subscriber and machines by 10. This results in $|S| = 1000$, $|E| = 50000$ and $|P| = 1000$ or $10000$ while each machine has a bandwidth of $2000$ or $5000$.

The performance of the two heuristics for the Homogeneous distribution and $|P| = |S| = 1000$ is shown in Figure 10. Overall, the two heuristics perform very close to the upper bound. The difference between GS and MKPBS is small, almost negligible, even though the latter has often a small

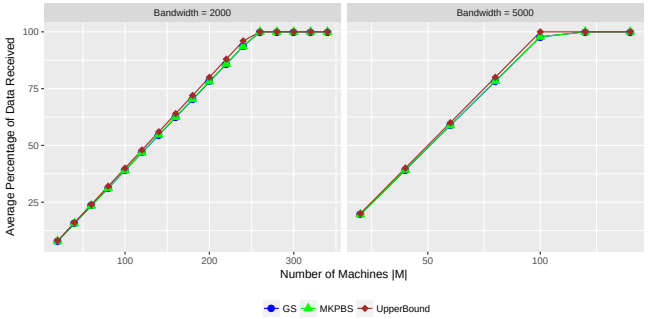


Figure 10. Average percentage of data received for GS and MKPBS for different value of bandwidth per machine. $|P| = 1000$, $|S| = 1000$ and each subscriber has 50 edges.

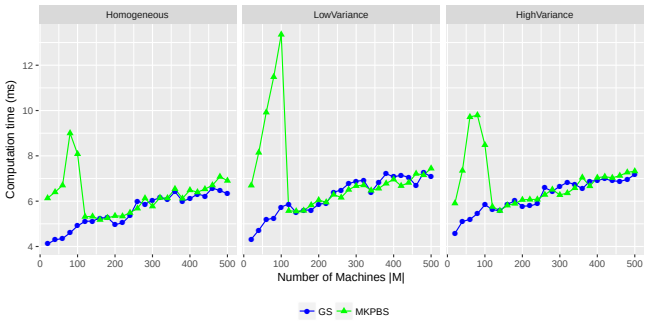


Figure 11. Average computation time in ms for GS and MKPBS. All machines have bandwidth equal to 5000, $|P| = 1000$, $|S| = 1000$ and each subscriber has 50 edges.

advantage, less than $0.1\%$ (50 edges). For $|P| = 10000$ and $|S| = 1000$, the performance gap slightly increases, to around $0.2\%$ (100 edges).

To evaluate scalability in terms of computation time, we consider two cases, $|P| = 1000$ and $|P| = 10000$. In the first case (Figure 11), the computation time is about two orders of magnitude larger than the computation time in Figure 7 (where everything is 10 times smaller and the total number of edges is 100 smaller), suggesting that $|E|$ is a dominant factor, in line with the complexity of the two heuristics. Note

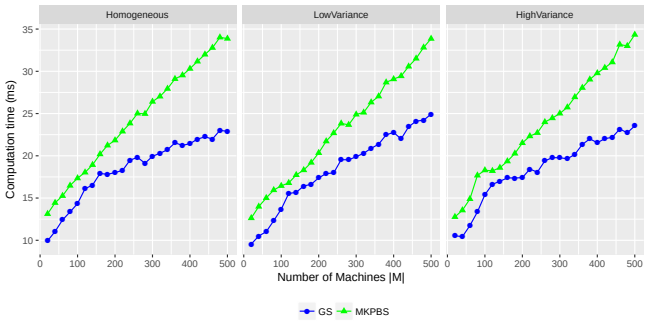


Figure 12. Average computation time in ms for GS and MKPBS. All machines have bandwidth equal to 5000, $|P| = 10000$, $|S| = 1000$ and each subscriber has 50 edges.

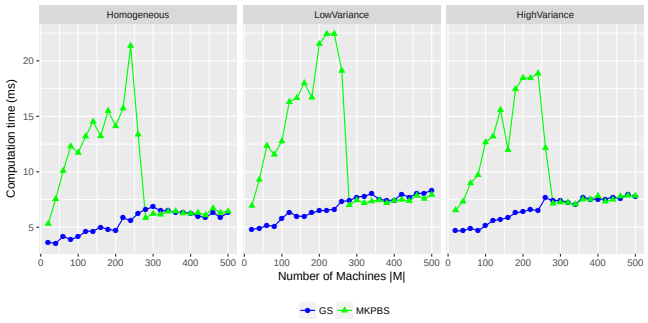


Figure 13. Average computation time in ms for GS and MKPBS. All machines have bandwidth equal to 2000, $|P| = 1000$, $|S| = 1000$ and each subscriber has 50 edges.

also that MKPBS keeps its unstable behaviour for a small number of machines but for a large number its computation time becomes similar to GS. In the second case, $|P| = 10000$ (Figure 12), the computation time difference with the results in Figure 8 (where, again, everything is 10 times smaller and the total number of edges is 100 smaller) is smaller than two orders of magnitude. This is because in Figure 8 $|E| < |P|$, which means that the impact of $|P|$ (and $|M|$) increases on computation time (cf. complexity analysis in Section IV). It is also interesting that GS in Figure 12 has a clear advantage, particularly as the number of machines increases.

Finally, we show the results for $|P| = |S|$ when the bandwidth per machine is 2000 (see Figure 13). The differences with the results of Figure 9 ($|P| = |S|$, bandwidth per machine set to 200) are of the same nature to the ones pointed above when comparing Figure 11 ($|P| = |S|$, bandwidth per machine set to 5000) and Figure 7 $|P| = |S|$, bandwidth per machine set to 500): computation times differ by about two orders of magnitude and MKPBS suffers from the same instability (until there are enough machines and hence enough available bandwidth). Furthermore, it is interesting that comparing Figure 13 with Figure 11 (whose only difference in terms of value settings is the value of bandwidth), it appears that only the computation time of MKPBS is affected.

*Summary:* Concluding this experimental section, it has been demonstrated that both GS and MKPBS achieve a near-optimal amount of received data compared to the optimal solution in small instances with Homogeneous distribution. In addition, both heuristics compute a solution very fast and scale well when the size of instances increase, while the amount of data received gets closer to the upper bound. In the context of these experiments, GS appears as a better option for instances of an important size; the very small loss in received data (compared with MKPBS) is compensated by a significantly faster and more stable execution time.

## VI. Conclusion

This paper investigates the problem of optimizing data sent using a pub/sub model and allocation of data links onto virtual machines. A formal problem, MaxDataRcvd, and an integer linear programming formulation have been provided. Two heuristics, a greedy one (GS) and one inspired by MKP (MKPBS) are introduced and experimentally studied. Both heuristics provide solutions of very good quality in the context they are tested. They also both exhibit short computation times and they appear to scale well as the problem size increases.

There are several ways to extend this paper. For example, the distribution of edges may vary (not all subscribers have the same number of edges) or different patterns for the input rates may be considered. One may also look into heterogeneous machines or additional aspects of network edges (e.g., related to security or different types of traffic). Scenarios where only an overall bandwidth is booked, which has to be repartitioned across machines may also be considered. Another direction is to define a separate function for the satisfaction of each subscriber (possibly depending on different types of data) not simply take the sum of data received.

## References


[1] A. Antonić, M. Marjanović, K. Pripužić, and I. P. Žarko, "A mobile crowd sensing ecosystem enabled by cupus: Cloud-based publish/subscribe middleware for the internet of things," *Future Generation Computer Systems*, vol. 56, pp. 607–622, 2016.

[2] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, Jun. 2003.

[3] V. Setty, R. Vitenberg, G. Kreitz, G. Urdaneta, and M. v. Steen, "Cost-effective resource allocation for deploying pub/sub on cloud," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, June 2014, pp. 555–566.

[4] Y. Sun, X. Qiao, B. Cheng, and J. Chen, "A low-delay, lightweight publish/subscribe architecture for delay-sensitive IoT services," in *2013 IEEE 20th International Conference on Web Services*, June 2013, pp. 179–186.

[5] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting iot platform requirements with open pub/sub solutions," *Annals of Telecommunications*, vol. 72, no. 1, pp. 41–52, Feb 2017.

[6] Y. Teranishi, R. Banno, and T. Akiyama, "Scalable and locality-aware distributed topic-based pub/sub messaging for iot," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–7.

[7] L. Guo, D. Zhang, G. Li, K.-L. Tan, and Z. Bao, "Location-aware pub/sub system: When continuous moving queries meet dynamic event streams," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD'15. New York, NY, USA: ACM, 2015, pp. 843–857.

[8] M. A. Shah and D. B. Kulkarni, "Storm pub-sub: High performance, scalable content based event matching system using storm," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, May 2015, pp. 585–590.

[9] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[10] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 713–728, 2005.

[11] S. Martello and P. Toth, "Heuristic algorithms for the multiple knapsack problem," *Computing*, vol. 27, no. 2, pp. 93–112, 1981.

[12] C. Cotta and J. M. Troya, "A hybrid genetic algorithm for the 0–1 multiple knapsack problem," in *Artificial neural nets and genetic algorithms*. Springer, 1998, pp. 250–254.

[13] D. Pisinger, "An exact algorithm for large multiple knapsack problems," *European Journal of Operational Research*, vol. 114, no. 3, pp. 528–541, 1999.

[14] A. S. Fukunaga, "A branch-and-bound algorithm for hard multiple knapsack problems," *Annals of Operations Research*, vol. 184, no. 1, pp. 97–119, 2011.

[15] "IBM ILOG CPLEX Optimizer." [Online]. Available: https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer

[16] "Plateforme Fédérative pour la Recherche en Informatique et Mathématiques," 2009. [Online]. Available: https://www.plafrim.fr/fr/accueil/