Non-clairvoyant Scheduling of Multiple Bag-of-tasks Applications

Henri Casanova[1], Matthieu Gallet[2,3], and Frédéric Vivien[3,1,2]

[1] University of Hawai'i at Mānoa, HI, USA
[2] École normale supérieure de Lyon, France
[3] INRIA, France

Abstract The bag-of-tasks application model, albeit simple, arises in many application domains and has received a lot of attention in the scheduling literature. Previous works propose either theoretically sound solutions that rely on unrealistic assumptions, or ad-hoc heuristics with no guarantees on performance. This work attempts to bridge this gap through the design of non-clairvoyant heuristics based on solid theoretical foundations. The performance achieved by these heuristics is studied via simulations in a view to comparing them both to previously proposed solutions and to theoretical upper bounds on achievable performance. Also, an interesting theoretical result in this work is that a straightforward *on-demand* heuristic delivers asymptotically optimal performance when the communications or the computations can be neglected.

1 Introduction

Bag-of-tasks (BoT) applications are parallel applications comprised of (many) independent and similar tasks. Although simple, this application model is typical of a large and relevant class of applications, including parameter sweep applications [1]. Due to their simple structure, many BoT applications are executed on volunteer computing platforms like BOINC [2]. Many studies are devoted to the scheduling of these applications on distributed platforms. On the one hand, a few studies [3,4,5,6,7] propose *theoretically sound* solutions. These studies consider either a single BoT application [3] or several [4,5,6,7]. They all take into account the costs, i.e., the overhead, of computations and communications. They assume that the scheduler has perfect knowledge of platform and application characteristics, that these characteristics do not change over time, and that the different tasks in a BoT application are perfectly identical. These assumptions rarely hold in practice. On the other hand, other studies propose *pragmatic* heuristics that are backed neither by a theoretical foundation, nor by a comparison to theoretical bounds on achievable performance. The corpus of proposed heuristics is large and diverse: some authors consider application characteristics to be perfectly known [8,9] and others consider them to be completely unknown [10]; some authors assume that all tasks in a BoT have the same characteristics [9] and others that they have different characteristics [10,11,8]; some authors assume perfect knowledge of platform characteristics [8,9], others

only partial knowledge [11,12], and some follow a knowledge-free approach [10]; some authors take into account communications [11] while others ignore them entirely [10,8,12]; while some authors consider a single BoT application [9], most consider several BoTs simultaneously [12,10,11,8]; finally, some authors consider failure-prone platforms [10,13] or input-data sharing between tasks [11].

Our goal is to design an approach that remedies the shortcomings of both the theoretical and the pragmatic previously proposed approaches. Our approach takes into account both computation and communication costs, and is applicable to the scheduling of several BoT applications whose characteristics are not perfectly known. Our main contributions are:

- A polynomial-time approximation scheme for the clairvoyant case;
- Non-clairvoyant heuristics that rely on historical information and that are based on the approximation scheme;
- A proof that the simple dynamic *on-demand* heuristic is asymptotically optimal when either communication costs or computation costs dominate;
- Exhaustive simulations to assess the quality of our heuristics.

The rest of the paper is organized as follows. We define the problem formally in Section 2. We study the clairvoyant case, that is, the theoretical case that assumes perfect knowledge, in Section 3. In Section 4 we consider the non-clairvoyant case where task execution times are not known beforehand. Section 5 presents our simulation results. Section 6 concludes the paper with a summary of results and future work directions. The missing proofs and additional simulation results can be found in a research report [14].

2 Models and Problem Definition

2.1 Platform Model

We consider a star-shaped distributed platform with a master processor P_0 connected to n worker processors $P_1, \ldots, P_n$. This platform is heterogeneous: worker P_i computes at speed s_i (in flop/s) and communicate with a bandwidth bw_i (in byte/s). For communications we use the bounded multi-port model [15]: a processor can be involved in an unbounded number of simultaneous communications, but the sum of the transfer rates of these communications is limited by the bandwidth of the network card.

2.2 Ideal Application Model

K bag-of-tasks applications are submitted to the system. The k-th BoT application, T_k, is a set of independent and perfectly identical tasks: the execution of each task requires an initial communication of size $V_{comm}(k)$ in bytes (to send the input data and, possibly, the program code) and its completion requires a computation volume $V_{comp}(k)$ in flops. We call the tasks of a BoT application *instances*. We use a linear cost model for both computations and communications: it takes $V_{comm}(k)/b$ seconds to send an instance of application T_k with

a bandwidth b, and $V_{comp}(k)/s_i$ seconds for worker P_i to process it. We allow the overlap of computations and communications: a worker can process an instance while retrieving another instance from the master. The master does not participate in the work. Like most previous works we neglect the transmission of output data back to the master.

2.3 Performance Objective

A traditional scheduling objective is the minimization of the overall execution time, or makespan. Since each application comprises a large number of instances, instead of targeting makespan minimization we focus on maximizing the average number of instances processed per time-unit, or *throughput*. This work is thus in the context of *steady-state* scheduling [16]. Note that the steady-state approach often allows to design asymptotically optimal schedules for makespan minimization, while makespan minimization itself is NP-complete. We use $\rho^{(k)}$ to denote the throughput of application T_k, and $\rho_i^{(k)}$ the contribution of worker P_i to this throughput. Although our goal is to maximize the overall throughput, i.e., the sum of the throughputs of the applications, we allow applications to have different priorities. π_k denotes the priority of T_k and we enforce the constraint that the throughput of an application is proportional to its priority: $\rho^{(k)}/\pi_k = \rho^{(1)}/\pi_1$.

2.4 Realistic Probabilistic Application Model

Section 2.2 describes the application model used in existing theoretical studies. This model assumes that instances of a BoT application have perfectly identical characteristics, which rarely holds in practice. Instances of a same application typically share some common features because they correspond to execution of the same program. However, their characteristics often differ, e.g., in the case when the program's execution time depends on the input data. To take these variations into account, we assume that for each BoT application there exist two probability distributions. One distribution describes the communication costs of the instances, and the other describes their computation costs. We use different distributions for each application, since different applications have different characteristics. Accounting for these differences should make it possible to produce better schedules. More formally, we consider two random variables (RV) for each application T_k:

- $X_{comm}^{(k)}$ is an RV that measures the communication cost of instances of T_k. $X_{comm}^{(k)}(u)$ denotes the realization of this RV for the u-th instance;
- $X_{comp}^{(k)}$ is an RV that measures the computation cost of instances of T_k. $X_{comp}^{(k)}(u)$ denotes the realization of this RV for the u-th instance;

We assume knowledge of lower and upper bounds on the values of these two RVs. Lower bounds are denoted $\min_{comm}^{(k)}$ and $\min_{comp}^{(k)}$, and upper bounds $\max_{comm}^{(k)}$ and $\max_{comp}^{(k)}$. The characteristics of the instances of application T_k are then fully described by the following joint probability distribution:

$$\forall c_1, d_1 \geq 0, \forall c_2 \geq c_1, \forall d_2 \geq d_1, \mathcal{P}\left(c_1 \leq X_{comp}^{(k)} \leq c_2; d_1 \leq X_{comm}^{(k)} \leq d_2\right).$$

In general, RVs $X_{comm}^{(k)}$ and $X_{comp}^{(k)}$ are not independent since computation times can depend on input data size. In practice, these distributions are not known but can be approximated based on historical observations.

3 The Clairvoyant Case

In this section we study the *clairvoyant* case, i.e., the case in which the characteristics of each application instance are perfectly known. We first recall the optimal solution for the ideal application model (Section 3.1). Building on this result, we design an approximation algorithm for our probabilistic application model (Section 3.2). Results in this section form the basis of scheduling heuristics for the more realistic non-clairvoyant case.

3.1 Optimal Solution for the Ideal Application Model

The optimal solution for our ideal application model is given in [4] (which is, in fact, applicable to a more general problem). This solution is expressed as an optimal solution of Linear Program (1), which expresses the following necessary and sufficient constraints to guarantee a valid schedule:

- The total throughput of application T_k is the sum of the contributions of all processors (Constraint (1a));
- The throughput of T_k is proportional to its priority (Constraint (1b));
- A worker cannot exceed its processing capability: the sum, over all applications, of the times it devotes to instance processing per time-unit cannot exceed the duration of a time-unit (Constraint (1c));
- A worker cannot exceed its incoming bandwidth: the sum, over all applications, of the times it devotes per time-unit to receiving instances cannot exceed the duration of a time-unit (Constraint (1d));
- The master cannot exceed its outgoing bandwidth: the sum, over all workers and applications, of the times it devotes per time-unit to sending instances cannot exceed the duration of a time-unit (Constraint (1e)).

The objective is to maximize the overall throughput. This is equivalent to maximizing the throughput of one of the applications, e.g., T_1, since the throughputs of all applications are proportionally related. All variables in the linear program are rational, meaning that an optimal solution can be computed in polynomial time.

$$
\begin{cases}
\text{MAXIMIZE } \rho^{(1)} \text{ UNDER THE CONSTRAINTS} \\
\text{(1a)} \quad \forall k, 1 \leq k \leq K, \quad \sum_{1 \leq i < n} \rho_i^{(k)} = \rho^{(k)} \\
\text{(1b)} \quad \forall k, 2 \leq k \leq K, \quad \frac{\rho^{(k)}}{\pi_k} = \frac{\rho^{(1)}}{\pi_1} \\
\text{(1c)} \quad \forall 1 \leq i < n, \quad \sum_{1 \leq k \leq K} \rho_i^{(k)} \frac{V_{comp}(k)}{s_i} \leq 1 \\
\text{(1d)} \quad \forall 1 \leq i < n, \quad \sum_{1 \leq k \leq K} \rho_i^{(k)} \frac{V_{comm}(k)}{bw_i} \leq 1 \\
\text{(1e)} \quad \sum_{1 \leq i < n} \sum_{1 \leq k \leq K} \rho_i^{(k)} \frac{V_{comm}(k)}{bw_0} \leq 1
\end{cases}
\tag{1}
$$

3.2 Approximation Scheme for the Probabilistic Application Model

With the more realistic probabilistic application model, the different instances of a given application T_k no longer have the same communication and computation characteristics, but we assume that we know their distributions. The idea underlying our approximation scheme is to split each application into several *virtual* applications such that two instances of the same virtual application have communication and computation characteristics sufficiently similar that considering them to be identical will not have a significant impact on the performance of the schedule. The virtual applications are defined based on a parameter ε. If ρ^* is the throughput of an optimal solution, our approximation scheme will produce a solution whose throughput ρ is at least equal to $\rho^*/(1+\varepsilon)$.

For application T_k, let $\gamma_q^{(k)} = (1+\varepsilon)^q \min_{comm}^{(k)}$, for $0 \leq q \leq Q^{(k)}$, $\delta_r^{(k)} = (1+\varepsilon)^r \min_{comp}^{(k)}$, for $0 \leq r \leq R^{(k)}$, $Q^{(k)} = 1 + \left\lfloor \ln\left(\frac{\max_{comm}^{(k)}}{\min_{comm}^{(k)}}\right)/\ln(1+\varepsilon) \right\rfloor$, and $R^{(k)} = 1 + \left\lfloor \ln\left(\frac{\max_{comp}^{(k)}}{\min_{comp}^{(k)}}\right)/\ln(1+\varepsilon) \right\rfloor$. Let $I_{q,r}^{(k)}$ be the interval $\left[\gamma_q^{(k)}; \gamma_{q+1}^{(k)}\right] \times \left[\delta_r^{(k)}; \delta_{r+1}^{(k)}\right]$. An instance of application T_k belongs to interval $I_{q,r}^{(k)}$ if its communication cost is comprised between $\gamma_q^{(k)}$ and $\gamma_{q+1}^{(k)}$ and its computation cost is between $\delta_r^{(k)}$ and $\delta_{r+1}^{(k)}$. *Virtual application* $T_{k,q,r}$ is the subset of the instances of application T_k that belong to interval $I_{q,r}^{(k)}$. $\rho_{i,q,r}^{(k)}$ is the contribution of processor P_i to the throughput of the virtual application $T_{k,q,r}$. Figure 1 shows an example with 3 applications partitioned into virtual applications with $\varepsilon = 0.4$. Note that some virtual applications are empty, like $T_{1,1,3}$.

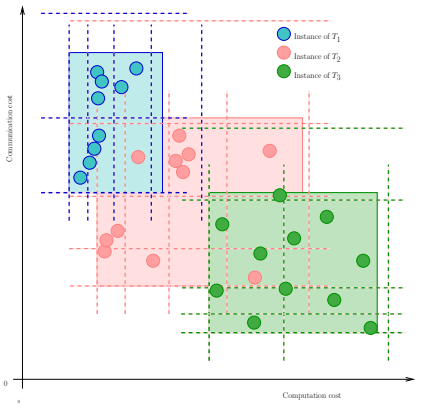

Figure 1. Example of partitioning into virtual applications ($\varepsilon = 0.4$).

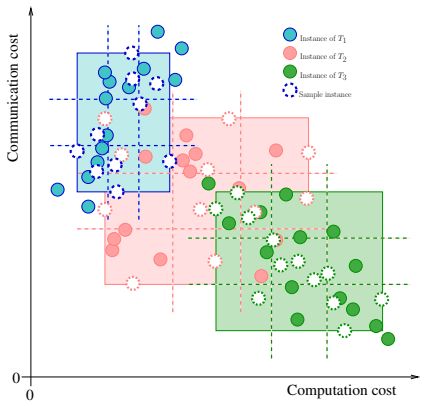

Figure 2. Partition of samples in 3×3 buckets.

The instances of a virtual application $T_{k,q,r}$ do not have the same communication costs and the same computation costs. However, by definition, all instances

of $T_{k,q,r}$ have a communication cost no greater than $\gamma_{q+1}^{(k)}$ and a computation cost no greater than $\delta_{r+1}^{(k)}$. We now apply the linear programming approach seen in Section 3.1. To do so, we follow a pessimistic approach: we approximate the communication and computation costs of all instances of $T_{k,q,r}$ by $\gamma_{q+1}^{(k)}$ and $\delta_{r+1}^{(k)}$. Finally, to adapt Linear Program (1) to the probabilistic application model, we need to account for the probability distributions of instance characteristics. More precisely, we denote by $p_{q,r}^{(k)}$ the probability that an instance of T_k belongs to $T_{k,q,r}$:

$$p_{q,r}^{(k)} = \mathcal{P}\left(\gamma_q^{(k)} \leq X_{comm}^{(k)} < \gamma_{q+1}^{(k)}; \delta_r^{(k)} \leq X_{comp}^{(k)} < \delta_{r+1}^{(k)}\right).$$

By construction, $\forall k, \sum_{q,r} p_{q,r}^{(k)} = 1$. If we randomly pick t instances of application T_k, we expect $t \cdot p_{q,r}^{(k)}$ of these instances to belong to virtual application $T_{k,q,r}$. We can then link the throughput of virtual application $T_{k,q,r}$, $\rho_{q,r}^{(k)}$, to the throughput of application T_k: $\rho_{q,r}^{(k)} = \rho^{(k)} p_{q,r}^{(k)}$. We obtain Linear program (2).

$$\begin{cases} \text{MAXIMIZE } \rho^{(1)} \text{ UNDER THE CONSTRAINTS} \\ \text{(2a)} \quad \forall k, 1 \leq k \leq K, \forall q < Q^{(k)}, \forall r < R^{(k)}, \sum_{1 \leq i < n} \rho_{i,q,r}^{(k)} = p_{q,r}^{(k)} \rho^{(k)} \\ \text{(2b)} \quad \forall k, 1 \leq k \leq K, \quad \frac{\rho^{(k)}}{\pi_k} = \frac{\rho^{(1)}}{\pi_1} \\ \text{(2c)} \quad \forall 1 \leq i < n, \quad \sum_{1 \leq k \leq K} \sum_{q < Q^{(k)}, r < R^{(k)}} \left(\rho_{i,q,r}^{(k)} \frac{\delta_{r+1}^{(k)}}{s_i}\right) \leq 1 \\ \text{(2d)} \quad \forall 1 \leq i < n, \quad \sum_{1 \leq k \leq K} \sum_{q < Q^{(k)}, r < R^{(k)}} \left(\rho_{i,q,r}^{(k)} \frac{\gamma_{q+1}^{(k)}}{bw_i}\right) \leq 1 \\ \text{(2e)} \quad \sum_{1 \leq i < n} \sum_{1 \leq k \leq K} \sum_{q < Q^{(k)}, r < R^{(k)}} \left(\rho_{i,q,r}^{(k)} \frac{\gamma_{q+1}^{(k)}}{bw_0}\right) \leq 1 \end{cases} \tag{2}$$

Theorem 1. *An optimal solution to Linear Program* (2) *achieves a throughput* ρ *no smaller than* $\rho^*/(1+\varepsilon)$*, where* ρ^* *is the optimal throughput.*

In the following we call ε-APPROX the schedule defined by an optimal solution of Linear Program (2) for a given value of ε. The careful reader may remark that we make the underlying assumption that our random picks are perfect: when we randomly pick t instances of application T_k, we assume that exactly $t \cdot p_{q,r}^{(k)}$ of them belong to virtual application $T_{k,q,r}$. While this is a theoretical weakness of our approximation scheme, simulation results Section 5 show that this weakness has no significant impact in practice.

4 The Non-Clairvoyant Case

In this section we consider the case in which some or all of the characteristics of an application instance are not known before the instance completes.

4.1 Performance of the On-Demand Heuristic

On-Demand is a simple dynamic heuristic: the master allocates a randomly chosen yet to be computed instance to the first worker that requests additional work. To take advantage of computation and communication overlap, each worker has a buffer that can contain $b \geq 1$ instances: while an instance is processed, a worker can ask for, receive, and store, at most b additional instances. Therefore, On-Demand does not make any distinction between the different processors, the different BoT applications, or the different instances of a BoT application. It turns out that On-Demand achieves asymptotically optimal performance when computation times dominate — i.e., when any computation takes more time than any communication and the output bandwidth of the master is never constraining — and when communication times dominate — i.e., when any communication takes more time than any computation. More formally, communication times dominate if

$$\frac{\min_{1\leq k\leq K} \min_{comm}^{(k)}}{\max_{0\leq i\leq n} bw_i} \geq \frac{\max_{1\leq k\leq K} \max_{comp}^{(k)}}{\min_{1\leq i\leq n} s_i},$$

and computation times dominate if

$$bw_0 \geq \sum_{i=1}^{n} bw_i \quad \text{and} \quad \frac{\min_{1\leq k\leq K} \min_{comp}^{(k)}}{\max_{1\leq i\leq n} s_i} \geq \frac{\max_{1\leq k\leq K} \max_{comm}^{(k)}}{\min_{0\leq i\leq n} bw_i}.$$

The additional constraint on the bandwidth of the master is required since we do not specify the behavior of communications in the presence of contention.

Theorem 2. *If either communication times or computation times dominate, and if each worker has a finite buffer, then* On-Demand *achieves asymptotically optimal performance.*

The constraint on the master's bandwidth can be relaxed. If processor P_i is always granted at least a fraction α_i of its input bandwidth when it requests work, then Theorem 2 still holds if

$$\frac{\min_{1\leq k\leq K} \min_{comp}^{(k)}}{\max_{1\leq i\leq n} s_i} \geq \frac{\max_{1\leq k\leq K} \max_{comm}^{(k)}}{\min_{0\leq i\leq n} \alpha_i bw_i}.$$

We conclude that On-Demand is asymptotically optimal for simple instances of the problem. However, practical heuristics are needed for the difficult cases, that is when neither the communications nor the computations can be neglected.

4.2 Practical Heuristics

In this section we adapt our ε-Approx algorithm to a non-clairvoyant framework. We assume that an instance's computation cost is not known until it

completes. However, we assume that communication costs are known. This assumption is reasonable because one should always be able to measure the size of the input data and of the program to be transferred.

Our practical heuristics are built on historical data. For each BoT application, we use a percentage of its instances (10% in our experiments) as a sample. We use the characteristics of this sample as if they were the characteristics of the whole BoT. For instance, we compute the lower and upper bounds on the communication costs of all instances based on this sample. Ideally, we would also be able to determine the probability $p_{q,r}^{(k)}$ that an instance of application T_k belongs to virtual application $T_{k,q,r}$, which is simply defined based on an interval (see Section 3.2). However, in a non-clairvoyant framework we have no estimates of instance computation costs and cannot compute this interval. Instead, we only consider intervals of the form $[\alpha;\beta] \times \left[\min_{comp}^{(k)}; \max_{comp}^{(k)}\right]$ to define virtual applications, i.e., simply not partitioning instances based on computation costs. The question is how to define α and β. We could conceivably pick a parameter ε and subdivide the range of communication sizes following the approach used for ε-Approx. However, recall that we have only a small sample of instances at our disposal. Subdividing each application into a large number of virtual applications would mean estimating the probabilities for each of these virtual application (the $p_{q,r}^{(k)}$'s) on even smaller sub-samples. Such estimates are likely to be of poor quality. For these reasons, we only subdivide each application into a small number v of virtual applications. Virtual application $T_{k,q,1}$ is then defined by the interval

$$\left[\min_{comm}^{(k)} \left(\frac{\max_{comm}^{(k)}}{\min_{comm}^{(k)}}\right)^{\frac{q-1}{v}}; \min_{comm}^{(k)} \left(\frac{\max_{comm}^{(k)}}{\min_{comm}^{(k)}}\right)^{\frac{q}{v}}\right] \times \left[\min_{comp}^{(k)}; \max_{comp}^{(k)}\right],$$

which is a simple geometric subdivision of $\left[\min_{comm}^{(k)}; \max_{comm}^{(k)}\right]$.

Once the virtual applications and their characteristics are defined, one can compute an optimal solution of Linear Program (2). To implement this solution we proceed as follows. The different instances of the different applications are scheduled in a random order. If the next instance to schedule is an instance of $T_{k,q,1}$, it is allocated to the available worker P_i that has the lowest ratio of $\rho_{i,q,1}^{(k)}$ to the number of instances of $T_{k,q,1}$ completed so far by P_i. We call the heuristic we have just described, which partitions each application into v virtual applications, LP_SAMP(v).

5 Experimental Results

5.1 Simulation Methodology

All algorithms are simulated using the SimGrid toolkit [17], and solutions of linear programs are computed by GLPK [18]. All source code is available at http://perso.ens-lyon.fr/matthieu.gallet/downloads/StochasticBOT.tar.gz.

Applications – Each simulation experiment is for $K = 3$ or $K = 4$ applications. All applications in an experiment have either 100, 1,000, or 5,000 instances. Based on [10,11,8], the characteristics of the instances of an application are sampled from uniform distributions. The communication cost v_{comm} of an instance of application T_k is in in interval $[\min_{comm}^{(k)}; \max_{comm}^{(k)}]$, where $\min_{comm}^{(k)}$ and $\max_{comm}^{(k)}$ are defined using two variables: *deviation*, which can take values 0.05, 0.35, and 0.65, and *magnitude*, which can take values 1, 2, 0.4, and 0.1. We have: $\min_{comm}^{(k)} = (1 - deviation)magnitude \times 10^5$ and $\max_{comm}^{(k)} = (1+deviation)magnitude \times 10^5$. In an experiment, all K applications use the same value of *deviation* but each application use a different value of *magnitude* among the first K possible values above. The computation cost v_{comp} of that same instance is then randomly and uniformly picked in the interval $[CCR \times (\phi v_{comm} + (1-\phi)\min_{comm}^{(k)}) \times 10^5; CCR \times (v_{comm} + (1-\phi)\max_{comm}^{(k)}) \times 10^5]$, where ϕ, the correlation factor, takes any value in $\{0; 0.5; 1\}$ and where CCR, the communication to computation ratio, takes any value in $\{0.05; 0.67; 1; 1.67; 20\}$. ϕ and CCR take the same value for all instances of all applications in an experiment. If $\phi = 0$, v_{comm} is not used in the definition of v_{comp}; if $\phi = 1$, v_{comp} is completely determined by v_{comm}. Finally, instances of all applications are shuffled before the whole set of instances is given as input to the scheduling algorithm.

Platforms – We consider platforms with 3, 5, 10, or 15 workers. The communication bandwidth (in byte/s) and the computation speed (in $10^5 \times$flop/sec) of each worker are randomly, uniformly, and independently picked in interval $[60,000; 140,000]$. The master's bandwidth is equal to 1, 5, 10, or 100 times the average bandwidth of the workers. For the first three values, the master's bandwidth may limit communications.

Heuristics – We use ROUND-ROBIN, which distributes instances to workers in a round-robin manner, and ON-DEMAND as references. We use LP_SAMP(v) for $v = 1$, 2, 4, and 8. These heuristics are all non-clairvoyant. We also use the clairvoyant 0.05-APPROX and 0.2-APPROX. All heuristics discard the same first 10% of the instances of each application. LP_SAMP(v) uses these instances as a sample. 0.05-APPROX and 0.2-APPROX use the set of all instances as a sample.

Performance metrics – For each heuristic, we compute the average throughput by dividing the total number of instances by the achieved makespan. We report on two types of average normalized throughputs: "normalized to best" and "normalized to upper bound." To compute the throughput normalized to best for a heuristic for a given simulation experiment, we divide the throughput of this heuristic by that of the heuristic that achieved the best throughput for this experiment. To compute the throughput normalized to upper bound for a given simulation experiment, we divide the throughput obtained by the heuristic by the upper bound computed according to Theorem 1, i.e., by multiplying by 1.05 the optimal throughput computed by Linear Program (2) for $\varepsilon = 0.05$. In both cases, the larger the relative throughput the better, with 1 being the max-

Heuristic	Normalized to best	Normalized to upper bound
On-Demand	0.870 ($\sigma = 0.108$, min = 0.638)	0.821 ($\sigma = 0.109$, min = 0.529)
Round-Robin	0.779 ($\sigma = 0.123$, min = 0.443)	0.736 ($\sigma = 0.126$, min = 0.371)
LP_samp(1)	0.971 ($\sigma = 0.036$, min = 0.692)	0.917 ($\sigma = 0.065$, min = 0.573)
LP_samp(2)	0.875 ($\sigma = 0.106$, min = 0.248)	0.829 ($\sigma = 0.122$, min = 0.200)
LP_samp(4)	0.819 ($\sigma = 0.130$, min = 0.213)	0.777 ($\sigma = 0.144$, min = 0.183)
LP_samp(8)	0.795 ($\sigma = 0.136$, min = 0.151)	0.754 ($\sigma = 0.149$, min = 0.139)
0.05-Approx	0.993 ($\sigma = 0.022$, min = 0.111)	0.937 ($\sigma = 0.055$, min = 0.097)
0.2-Approx	0.985 ($\sigma = 0.020$, min = 0.178)	0.930 ($\sigma = 0.051$, min = 0.148)

Table 1. Results averaged over 32,400 simulation scenarios. standard deviation σ and the minimum throughput are shown in parentheses.

imum possible value. We present normalized throughput results averaged over all simulation experiments.

5.2 Results

We have performed experiments for the 32,400 simulation scenarios defined by the parameter values given in the previous section. Results are shown in Table 1. The first observation is that the best results, for both metrics, are achieved by the two clairvoyant algorithms, 0.05-Approx and 0.2-Approx. 0.05-Approx achieves an average throughput normalized to best above 99%, meaning that is the best performing heuristic for most problem instances. Among the reference algorithms, not surprisingly, On-Demand achieves better and more stable performance than Round-Robin according to both metrics. LP_samp(1) achieves even better and more stable performance. In other words, our heuristic, just by differentiating the different BoT applications, leads to higher throughput. LP_samp(1) bridges half of the gap between the performance of On-Demand and the theoretical upper bound. Increasing v, i.e., the number of virtual applications, degrades performance. This is especially true for scheduling instances containing few instances per applications (see below). Furthermore, increasing v is only worthwhile if the virtual applications in a same application have significantly different CCRs. This is rarely the case in our experimental settings, hence the overall degradation of performance when v increases. For applications where the CCR is strongly correlated to the communication cost, increasing the number of virtual application does lead to significant increase in performance. We refer the reader to [14], a research report that contains full experimental results, and summarize salient results below:

Communication-to-computation ratio (CCR) – As predicted by Theorem 1, On-Demand returns very good schedules when the CCR is either very large (20) or very small (0.05), but it is then tied by LP_samp(1). Expectedly, the performance of On-Demand is worst for intermediate values of the CCR. The performance of LP_samp(v) heuristics is mostly independent of the CCR.

Number of instances per application – Unlike On-Demand, LP_SAMP(v) heuristics are sensitive to the number of instances. However, even for 100-instance experiments, LP_SAMP(1) achieves on average better throughputs than On-Demand (95.1% vs. 87.9%, for throughputs normalized to best). But for 100 instances increasing the number of virtual applications has a strong negative impact (since the sub-samples are far too small to be meaningful). Moving from 100 to 1,000 instances has a limited impact on LP_SAMP(1) but dramatically improves the performance of the LP_SAMP($v > 1$) heuristics.

Correlation factor – The performance of all LP_SAMP(v) heuristics increases with the correlation factor ϕ, but LP_SAMP(1) remains the best heuristic.

Running times – Solving a linear program for 20 workers and 100 different applications made of $5,000$ instances each leads to a running time of 2 seconds for parsing all data and solving the linear program, and an overall simulation time of 22 seconds.

6 Conclusion and Perspectives

We have studied the problem of scheduling multiple bag-of-tasks (BoT) applications when the different tasks in a bag do not have the same characteristics, but have characteristics that follow an unknown distribution. For the theoretical clairvoyant case (computation and communication costs are known beforehand), we have designed an approximation algorithm. Based on this algorithm we have designed a family of heuristics for the practical non-clairvoyant case (only communication costs are known). These heuristics, called LP_SAMP(v), infer computation costs from the execution of past instances, and partition application instances into v subsets which we have called virtual applications. We have proved that the simple On-Demand heuristic is asymptotically optimal when either communication or computation dominates. Through simulation, we have demonstrated that the LP_SAMP(1) heuristic always delivers good performance, and achieves significantly better performance than On-Demand in the most difficult cases (e.g., communication-to-computation ratio (CCR) close to 1). In other words, LP_SAMP(1), just by differentiating the different BoT applications, leads to better performance, even in the very difficult non-clairvoyant context and with BoT applications with few instances. This proves, once again, the power of statically defined heuristics. The subdivision of an application into a larger number of virtual applications, i.e., LP_SAMP($v > 1$), is only worthwhile for applications that have large numbers of instances and with a CCR that depends on communication costs.

An interesting future direction would be to adapt our LP_SAMP(v) heuristics to the *online* case, i.e., when applications are released at different times. Other interesting directions include the automatic definition of the optimal number of virtual applications for each application, and the development of techniques to accommodate non-dedicated processors.

Acknowledgments: This work is partially supported by the European Commission's Marie Curie Fellowship IOF #221002.

References

1. Casanova, H., Berman, F.: Grid'2002. In Berman, F., Fox, G., Hey, T., eds.: Parameter sweeps on the grid with APST, Wiley (2002)
2. Anderson, D.: BOINC: A system for public-resource computing and storage. In: GRID'04: 5th IEEE/ACM Int. Workshop on Grid Computing. (2004) 4–10
3. Banino, C., Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Robert, Y.: Scheduling strategies for master-slave tasking on heterogeneous processor platforms. IEEE Trans. Parallel Distributed Systems **15**(4) (2004) 319–330
4. Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Marchal, L., Robert, Y.: Centralized versus distributed schedulers for multiple bag-of-task applications. IEEE Trans. Parallel Distributed Systems **19**(5) (2008) 698–709
5. Benoit, A., Marchal, L., Pineau, J.F., Robert, Y., Vivien, F.: Scheduling concurrent bag-of-tasks applications on heterogeneous platforms. IEEE Transactions on Computers **59**(2) (2010) 202–217
6. Bertin, R., Legrand, A., Touati, C.: Toward a fully decentralized algorithm for multiple bag-of-tasks application scheduling on grids. In: Grid Computing, 9th IEEE/ACM International Conference on. (2008) 118–125
7. Legrand, A., Touati, C.: Non-cooperative scheduling of multiple bag-of-task applications. In: INFOCOM 2007. (2007) 427–435
8. Sulistio, A., Buyya, R.: A time optimization algorithm for scheduling bag-of-task applications in auction-based proportional share systems. In: Computer Architecture and High Performance Computing (SBAC-PAD 2005). (2005) 235–242
9. da Silva, F.A., Carvalho, S., Hruschka, E.R.: A scheduling algorithm for running bag-of-tasks data mining applications on the grid. In: Euro-Par 2004 Parallel Processing. Volume 3149 of Lecture Notes in Computer Science. (2004) 254–262
10. Anglano, C., Canonico, M.: Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach. In: IPDPS 2008. (2008)
11. Lee, Y.C., Zomaya, A.Y.: Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. Computers, IEEE Transactions on **56**(6) (2007) 815–825
12. Iosup, A., Sonmez, O., Anoep, S., Epema, D.: The performance of bags-of-tasks in large-scale distributed systems. In: HPDC '08, ACM (2008) 97–108
13. Kondo, D., Chien, A., Casanova, H.: Scheduling Task Parallel Applications for Rapid Application Turnaround on Enterprise Desktop Grids. Journal of Grid Computing **5**(4) (2007) 379–405
14. Casanova, H., Gallet, M., Vivien, F.: Non-clairvoyant scheduling of multiple bag-of-tasks applications. Research report 2010-08, LIP - ENS Lyon (2010)
15. Hong, B., Prasanna, V.: Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. In: IPDPS'04, IEEE CS Press (2004)
16. Beaumont, O., Legrand, A. Marchal, L., Robert, Y.: Steady-state scheduling on heterogeneous clusters. Int. J. of Found. of Computer Science **16**(2) (2005) 163–194
17. Casanova, H., Legrand, A., Quinson, M.: SimGrid: a generic framework for large-scale distributed experimentations. In: 10th IEEE International Conference On Computer Modeling and Simulation, IEEE CS Press (2008) 126–131
18. Makhorin, A.: GLPK (GNU Linear Programming Kit). http://www.gnu.org/software/glpk/ (2009)