J Grid Computing (2016) 14:347–358
DOI 10.1007/s10723-015-9347-6

A Set of Successive Job Allocation Models in Distributed Computing Infrastructures

Gábor Bacsó · Tamás Kis · Ádám Visegrádi · Attila Kertész · Zsolt Németh

Received: 20 March 2015 / Accepted: 12 August 2015 / Published online: 17 October 2015

Abstract The growing number of scientific computation-intensive applications calls for an efficient utilization of large-scale, potentially interoperable distributed infrastructures. Parameter sweep applications represent a large body of workflows. While the principle of workflows is easy to conceive, their execution is very complex and no universally accepted solution exists. In this paper we focus on the resource allocation challenges of parameter study jobs in distributed computing infrastructures. To cope with this NP-hard problem and the high uncertainty present in these systems, we propose a series of job allocation models that helps refining and simplifying the problem complexity. In this way we present some special cases that are polynomial and show how more complex scenarios can be reduced to these models. It is known from practice that a small number of job sizes improves the result of job allocation, therefore we state a hypothesis relying on this fact in one of our models. Unfortunately, the reduction of the general problem (using K-means clustering) did not help, and thus the hypothesis has proved to be false. In the future, we shall look for clustering techniques which fit this goal better.

Keywords Distributed computing infrastructures · Job allocation · Parameter sweep applications

Z. Németh (✉) · G. Bacsó · T. Kis · Á. Visegrádi · A. Kertész
MTA SZTAKI Computer and Automation Research Institute, P.O. Box 63, 1518 Budapest, Hungary
e-mail: nemeth.zsolt@sztaki.mta.hu

G. Bacsó
e-mail: bacso.gabor@sztaki.mta.hu

T. Kis
e-mail: kis.tamas@sztaki.mta.hu

Á. Visegrádi
e-mail: visegradi.adam@sztaki.mta.hu

A. Kertész
e-mail: kertesz.attila@sztaki.mta.hu

1 Introduction

While the principle of workflow applications [1] is easy to conceive, executing such an application is a very complex task and no universally accepted solution exists. The workflow itself is a logic description how tasks are related to each other and what are the conditions (requirements) of their execution, hence the name ”abstract workflow”. During execution such *abstract workflows* are translated into *concrete workflows* where all such conditions and requirements are analysed, evaluated (or fulfilled if necessary) by an automated mechanism. Parameter sweep applications or a set of parameter study jobs represent a specific and commonly used class of workflows, where ordering of task executions is irrelevant. In a slightly different context such applications are also called as embarrassingly parallel applications or bag-of-tasks applications. One of the most important requirements a task certainly has is its resource needs.

All computational tasks[1] must be mapped onto some computing resources so that the task can be realised and the necessary computations executed. During this process the resources should be selected so that the overall execution meets some criteria of optimum, typically the shortest possible execution time.

Such problems are known as *allocation* problems, one of the fundamental ones of parallel and distributed computing. Hence, they are extremely well studied; on the other hand they are known to be NP-hard [12] therefore, no exact and quick solution can be provided for them. Most of the approaches are based on some heuristics – and there is a very wide range of possible heuristics as *Load Balancing* [13], *Shortest Processing Time First* [15, 16], *Largest Processing Time First* [14], but more importantly they are typically aimed at a particular problem or a particular class of problems.

This paper follows and advances the pattern of [2] which was aimed at investigating how allocation based on elementary probabilistic theory can be efficient if supported by predictions distilled from historic trace data. While the paper has proven its assumption that these very basic probabilistic approaches can indeed produce effective allocation, it also revealed that subtle details can significantly make the allocation problem more complex and requiring more elaborated solutions. The present paper is aimed at showing several different approaches which are all relevant. For some of them, we present the answer, while we illustrate the difficulties for others. Finally, we concentrate on one specific approach and state a hypothesis concerning it. This hypothesis has proved to be false. We shall treat the conclusions in the last section.

The contribution of this paper is its perspective on the details of the task allocation problem and how they are revealed and analysed in successive models. Starting from the most basic deterministic (and practically unrealistic) one which is extremely easy to study, we add details in successive models and discuss the new features. It is interesting to note that the solution of some more complex models can be reduced to the first basic model. The models towards the end of the list are more realistic and in alignment with practical allocation scenarios and problems. We also provide an analysis of an approximation method based on clustering heuristics.

[1] "Tasks" are used in the workflow terminology whereas "jobs" are of the scheduling community. We use "jobs" in the rest of the paper.

The remainder of the paper is as follows: Section 2 summarizes related work. Section 3 introduces the assumptions of the model, i.e. an abstract view of the allocation scenario. Sections 4 through Section 9.1 describe the successive models; Section 4: the basic deterministic model; Section 5: different processing speed of nodes modeled as a linear approximation; Section 6: different processing speed of nodes modeled as a general function; Section 7: reliability of nodes modeled as constants; Section 8: probabilistic model of reliability of nodes; and in Section 9 the general makespan problem. The latter model is analysed and evaluated by experiments in Section 10 and the paper is concluded in Section 11.

2 Related Work

Job allocation and scheduling on a multiprocessor system has been studied for more than 30 years and most of its variants are known to be *NP-hard* [12].

Scheduling in Grid systems, which is one of the tasks of Grid resource managers, become even more complicated with multi-organizational shared resources, therefore Grid scheduling is also NP-hard [3]. Schwiegelshohn et. al. [6] showed that the performance of Garey and Graham's list scheduling algorithm is significantly worse in Grids than in general multiprocessor systems. They proposed a Grid scheduling algorithm using the "job stealing" approach that guarantees a competitive factor of 5. (This factor is the ratio between the performance of a specific online algorithm and the best possible offline solution. In case of an NP-hard problem, this solution may be difficult to obtain.) The basic idea of this approach is if there are no jobs available in the lists of a machine, this machine starts to use the list of a neighboring machine involving local communication only.

In order to achieve better scheduling, in some works approximates and run time estimates are used, e.g. [10]. On the other hand, the inaccuracy of these estimates is a perennial problem mentioned in the literature, and even if users are required to provide these values, there is no substantial improvement in the overall average accuracy [4]. Ramirez-Alcaraz et al. [5] have analyzed different Grid allocation strategies depending on the type and amount of information they require, and they found that information about users'

runtime estimate and local schedules does not help to significantly improve the outcome of the allocation strategies. They concluded that quite simple schedulers with minimal information requirements can provide good performance. Practice seems to adapt to these findings, because too complex, sophisticated scheduling algorithms are rarely used in Grid brokers.

Cirne et al. [23] developed Workqueue with replication (WQR) that keeps computing resources busy so that if a node runs out of work and would become idle, a copy of an unfinished task is assigned to it. In such a way a second chance is given for a faster execution: the two copies of the same task compete. The approach does not need any information about the tasks or the resources and can dampen the effect of dynamic loads and improve execution time by wasting some (controlled limit) resources. Following this idea Da Silva et al. in [24] also raise the question of methods that require information compared to WQR. WQR is however, iterative as opposed to our single decision making for allocation.

Casanova et al. [22] focused on the very same problem of scheduling parameter sweep applications on Grids, with particular attention to file transfers and network performance. Also, their motivation is in alignment with most of the papers in this area: inaccurate predictions can largely mislead scheduling. Their approach is modifying existing heuristics so that they are adaptive in a dynamic heterogeneous environment. The core of the scheduling is a Gantt chart that is created and updated periodically and keeps track of job and resource assignments. Assignments are governed by a heuristics called *sufferage* where a task is assigned to a host if the task would "suffer" the most if done otherwise; "suffering" is expressed as the difference between the best and the second best minimum completion times. A considerable complexity is added in this model by taking into account file transfer and communication costs; this lead to modify the definition of the sufferage algorithm.

Maheswaran et al. [21] addresses the issue of mapping independent tasks onto heterogeneous computing systems. They apply heuristics aiming at optimizing for *throughput*, i.e. increasing the finished task per time unit ratio. It considers Minimum Completion Time (MCT), Minimum Execution Time (MET), Switching Algorithm (switches between MCT and MET) and k-percent Best (MCT on a subset of resources) as on-line heuristics whereas Min-Min, Max-Min [11] and Sufferage [21] for batch mode ones. Min-Min and Max-Min roughly correspond to MCT and MET of on-line algorithms. Their analysis revealed that batch scheduling can outperform on-line scheduling for large number of tasks, on the other hand some on-line scheduling may have lower computation time. Within the batch scheduling class, Min-Min and Sufferage were proven to be superior.

Menascé at al. [20] introduce further static scheduling by combining an envelope (a selection of tasks and resources to be considered) and a heuristics. In the paper they derived 6 static scheduling methods by crossing three task selection and two processor selection heuristics; these were compared with three dynamic ones. They claim the superiority of static methods: these are executed rarely hence, more sophisticated algorithms can be realized without time penalty. All these methods are based on task priorities whereas, in our case we do not differentiate between task priorities.

Oprescu et al. [7] propose a budget constraint-based resource selection approach for Cloud applications. In this work they present a budget-constrained scheduler called BaTS, which can schedule bags of tasks onto multiple clouds with different CPU performance and cost, minimizing completion time with maximized budget. Their scheduler learns to estimate task completion times at run time. GridBot [19] represents an approach for execution of bags-of-tasks on multiple Grids, clusters, and volunteer computing Grids. It has a Workload Manager component that is responsible for brokering among these environments, which is similar to our approach, but they focus on tasks more suitable for volunteer Grids.

Hirales-Carbajal et al. [17] present an experimental study of 22 deterministic non-preemptive multiple workflow scheduling strategies in Grids. While their objective is to schedule and execute the whole workflow, and minimize its makespan, we restrict ourselves to parameter study jobs of such workflows.

3 Elements of the Models and the Assumptions

To provide a general solution, we introduce an abstract model for the computing infrastructure and the tasks, independently from any actual physical realization. *Jobs* are the computational tasks, roughly speaking the "programs to be executed" that are independent from

each other. Jobs start, do some computation and termi nate with a result or abort with a failure; beyond these three stages of behavior it is indifferent what compu tation they actually carry out or what their internal structures are. We assume a set of k jobs where k is reasonably large. The time between the start and the termination or abort of the job is called the *execution time*.

Resources are the units of a DCI. This is an abstract notion in the sense that it may be physically realized as a single processor (computer) or a set of processors (a cluster) nevertheless, from scheduling or mapping point of view, resources are the smallest units that can be observed or controlled. Obviously, resources may contain smaller (finer-grained) computing elements like cores of a processor or processors of a cluster. Albeit some models will take these into account, they are managed by an operating system or a middleware and are assumed hidden from the scheduling or mapping agent. We assume a set of n resources. Each resource may have a pre-load: jobs that are submitted by other parties thus, adding some delay to the execution. We define c_i as the pre-load of resource i.

Generally, mapping is an assignment of jobs to resources so that it conforms certain constraints (e.g. capacities, functional properties, etc.) and fulfils some criteria of optimum (such as running time, communication volume, used resources, consumed energy, etc.) In the context of the paper we restrict ourselves to a mapping that is aimed at minimizing the makespan A of the job execution, i.e. minimize the time that takes the execution of the job that finishes last. The notation of our model is summarized in Table 1.

Table 1 Summary of parameters and quantities used in the algorithms

$n \in \mathbb{Z}$	number of all available resources
$k \in \mathbb{Z}$	number of jobs to be allocated
$k_i \in \mathbb{Z}$	number of allocated jobs on resource i
$c_i \in \mathbb{R}$	delay (waiting) caused by other jobs (pre-load) on resource i
$\mathbf{x} = (k_1, k_2, \ldots k_n)$	k_i: number of jobs allocated on resource i
J	expected value of t_j, $\mathbb{E}(t_j)$
R_i	running time (pre-load + own jobs) on resource i
A	makespan (total execution time)

4 Model 1 – Basic, Deterministic Model for Allocation

The target is to allocate k jobs to some of n resources representing the nodes of a heterogeneous DCI (i.e. a Grid or a Cloud) so that each resource has a certain pre-load c_i (jobs belonging to different applications).

In Model 1 it is assumed that the jobs to be allocated have *unit* execution times thus, k_i jobs take k_i time and c_i is equivalent to a delay of c_i time units. Resource i finishes its operation after $R_i = c_i + k_i$ time units. (In fact, $R_i = R_i(\mathbf{x})$, the function of the execution times for a given job allocation $\mathbf{x}$ (see Table 1). We define turnaround time $A(\mathbf{x}) = \max_{i,k_i \neq 0} (R_i(\mathbf{x}))$. The goal is to minimize A, i.e. find an allocation $\mathbf{x}$ so that $A = \min_{\mathbf{x}} A(\mathbf{x})$.

Remark 1 The restriction $k_i \neq 0$ is necessary in the formula of maximum since the resources without new jobs allocated there are considered to be non-existing.

Solution 1 An algorithmic solution for Model 1 can be found in [2]. The algorithm can be applied for any non-negative real numbers c_i (later we will use this fact, in Section 8.2). Several quick algorithms exist for this simple problem. We may consider both the problem and the solution as a building block (a black box, independently of the solution used.) The importance of these blocks (for Model 1 and also for Model 2) will be clear in Section 8.2, incorporating their application.

5 Model 2 – A Modified Function for the Running Time

In practice, often the purely additive formula for the running time ($R_i = k_i + c_i$) is not valid. One of the alternative formulae is the following.

(Model 2) The resources have their own speed β_i. Thus the running time on resource i will be calculated as $R_i = \beta_i k_i + c_i$. Here we define a general notion in connection with optimization problems.

Definition 1 For a function $A(\mathbf{x})$, the question whether there exists an $\mathbf{x}$ with $A(\mathbf{x}) \leq B$ for an arbitrary B we call the *decision problem* in connection with the minimization problem of function $A(\mathbf{x})$.

Here B can be chosen arbitrarily. In the same sense, we may speak on the decision problem in connection with a maximization algorithm.

The decision version of the deterministic algorithm, developed in our former paper [2] can easily solve the task in Model 2 as well. In the next section we generalize this result, too.

6 Model 3 – Generalization of Model 2

As we shall see, the functions R_i of Model 2 are only the special cases of a class of functions for which the makespan problem can be easily solved. However, we emphasize that the whole Model 3 is valid in case of equal job sizes only.

Let us suppose we have a function f_i for each $i = 1, ..., n$ (that is, for each resource). The meaning of f_i is that j jobs will be finished in (exactly!) $f_i(j)$ time on the ith machine. A good example is $f_i(x) = R_i(x)$, where R_i is one of the functions in the previous section.

We are able to use the method below if the ith function satisfies the following condition: For each i, f_i is monotone and it can be calculated by a quick algorithm.

Proposition 1 1. *If the condition above is fulfilled for the functions f_i then, given a bound, we are able to calculate the maximum number of jobs, allocatable to a a given resource without the violation of the time bound.*

2. *Consequently, for such running time functions, the minimum makespan problem can be quickly solved.*

Proof

1. Let the given time bound be denoted by B. Putting j jobs on the ith resource, it does not violate this bound if and only if $f_i(j) \leq B$. By the condition, we easily obtain the maximal $j = j(i)$ for which $f_i(j) \leq B$. Summarizing, for every resource i we know, how many jobs we are allowed to put there, considering the time bound – the answer is $j(i)$. If $\sum_{i=1}^{n} j(i) \leq k$ then there exists an allocation $\mathbf{x}$ with $A(\mathbf{x}) \leq B$, otherwise not.
2. This means that the *decision problem* in connection with our minimization problem can be quickly solved, consequently, so is the minimization problem itself.

□

Model 3 may have many applications, because of the large diversity of resources and running time functions, occurring in practice. The makespan problem can be solved always when the (very weak) conditions of Proposition 1 are valid. For instance, the j^{th} job gets the output of each preceding job thus, its running time increases by j. The formula for the function

$$R_i = 1 + 2 + ... + (k_i - 1) = \frac{(k_i - 1)k_i}{2}$$

This function satisfies the condition above. On the other hand, Model 3 shows, that sometimes even strangely abstract models can help.

7 Model 4 – Reliability with Constant Penalty

In this model we try to capture the case when some jobs on some resources abort and we have a statistics for each given resource on the rate of aborts or failures. This rate, i.e. reliability, can be different for different resources. Thus, for the ith resource we have a probability p_i for the successful running of a job allocated there. Unfortunately, the consequence of an abort cannot be described in any allocation model (used here), because all these models need a decision made in a given moment, in accordance with the name "allocation" (see Introduction). The consequences may involve a significant loss of time in connection with the job aborted, rescheduling the job, and so on.

Despite these difficulties, managing the mathematical problem of aborts cannot be avoided. There are many ways of formalizing this phenomenon. One of the solutions is to associate a penalty with an abort – a constant time penalty T. For the determination of T, experience is the only possibility. This is the roughest model, thus we will solve a more general one in the next section. one can easily treat it as we will see in Section 8.2.

8 Model 5 – Reliability with Penalty Distribution

This model is a refinement of Model 4, and a little bit more realistic. Whenever a job is aborted due to some failure, its re-enactment takes more time and resources, represented as time loss or penalty. Suppose, the experience on the abort phenomenon can be coded in a probability distribution of the time loss. It is known that the number of aborts has a binomial distribution [18] (we shall apply this fact in Proposition 2), but the authors did not find any realistic loss distribution yet. One may expect that the correlation of these two random variables, is small (often 0). However, as it will be treated below, one can partially manage the problem, using only a realistic penalty distribution.

8.1 Model 5' – Zero Correlation

Model 5' is a special case of Model 5 above where the two random variables – the number of successful jobs and the loss of time – are independent, i.e. their correlation is 0.

The solution to this problem is reduced to the one presented in Section 4, i.e. the elementary solution of a simple deterministic problem, taken as a black box is incorporated into this more sophisticated approach, as introduced in details in the following subsection.

8.2 A Method for Solving a Non-deterministic Problem by Deterministic Tools

The key step of this method is that we restrict ourselves to one information on the probability distributions of all the random variables, occurring in the problem – namely their expected value. Furthermore, we use the obvious property of the expected value being additive. This yields a possibility to reduce the stochastic problem to the task described in the first model.

An Example – Reduction of Model 4 to Model 2 Given k jobs with unit runtime and n resources. The pre-load on resource i is c_i ($c_1 \leq c_2 \leq \ldots \leq c_n$). After allocation, k_i jobs will run on resource i and each of them will be finished with probability p_i. In case of successful execution the runtime of the job will be a unit, otherwise we consider that its runtime will be T where T is the time penalty. The runtime on resource i is a random variable t_i, with expected value R_i. The task is to minimize makespan $A(\mathbf{x})$ over all assignments $\mathbf{x}$.

The number of jobs successfully finished on resource i is denoted by Y_i (it is a random variable). With this notation

$$t_i = Y_i + (k_i - Y_i)T + c_i \tag{1}$$

Proposition 2 *For the expected value $E(Y_i)$,*

$$E(Y_i) = p_i k_i$$

Proof Y_i has a binomial distribution (see above), thus we may use this well-known fact in probability theory [18]. □

This implies

$$R_i = q_i k_i + c_i \tag{2}$$

where $q_i := T-(T-1)p_i$ (2) means that we can simulate this non-deterministic model by the deterministic Model 2.

A new Example – Reduction of Model 5' to Model 2 The only necessary modification of the calculation in the former section is the following. In (1) t_i, the product of the random variables Y_i and T appears. Their independence implies that the expected value can be calculated in the same way and the previously introduced method can be applied.

Remark 2 The same reduction is impossible without the assumption of independence.

9 Model 6 – the General Makespan Problem

As we know, the initial deterministic problem with arbitrary job sizes and zero pre-loads is NP-hard – it is called the Makespan problem in the literature and its decision problem in connection with it is the same as that of the famous Bin Packing problem [8]. (This problem was treated in Section 5.) The approaches below are attempts to cope with the Makespan problem, always in the sense that we solve an easier

problem, and thus we obtain a solution, 'good enough' for the Makespan Problem.

9.1 Clustering Method: Reduction to a Constant Number of Job Sizes

Though in a real situation we find a lot of job sizes, considering a system as a non-deterministic process, we are able to make groups such that, within a given group, the job sizes can be viewed as the values of a random variable with small deviation. For a constant number k of job sizes, the Makespan problem can be solved better in practice (and, from a theoretical point of view, there exist polynomial algorithms for this case [8]. Suppose the number of groups is small, then, applying "the method of mean", described in [2], we obtain a good estimation of the real system.

Remark 3 In Section 10 we shall describe tests in connection with the clustering method.

9.2 Approximation Method

The solution of the approximation problem means that we do not search for the minimum makespan M but we want to have a solution with makespan at most $(1+\epsilon)M$. For every ϵ, there is polynomial algorithm but it is not usable in practice. The reason is that in the exponent of time estimation we can find (roughly) $1/\epsilon$ [8]. However, for $\epsilon = 1/3$, a quick algorithm, Longest Processing Time First (LPT) is known, classical already (can be found in the well-known paper of Graham [14], also containing the List Scheduling Algorithm). A further advantage of LPT is that it is a so-called quasi-online algorithm, namely, after making a preorder, it works as an online algorithm. Here we emphasize that our model is static in the sense that the whole set of the jobs to be executed is given in advance.

Remark 4 Once we combined Clustering and Approximation, we arrive to a new model. Seemingly, this model is too sophisticated. However, the mathematical methods of scheduling can often solve such combinations relatively easily. If solved, we could obtain better results than both of the two former models. In the next section, for the models above we shall use both zero and general pre-loads.

10 Evaluations of the Clustering Method Combined with LPT

We have also performed measurements based on the Clustering Method where the LPT algorithm is used to allocate jobs on the distributed infrastructure. This algorithm works with – a preferably small number of – distinct job sizes (in terms of execution time). In the former (discrete) case, the less the number of possible job sizes are, the better LPT performs [9]. In the extremal case when there is only a single possible job size, LPT trivially achieves optimal results [2].

In a real world scenario, jobs may be of any size, in which case the LPT algorithm tends to produce makespans closer to $(1 + 1/3)M$.[14] We may try to reduce the number of possible job sizes to a small given number by using K-means clustering with different values for K. Each job can then be assigned to a representative element, the mean in its cluster, then the LPT algorithm can allocate these jobs based on their representatives. Our hypothesis is the following:

Hypothesis As these representatives have only a small number (K) of possible values, the LPT algorithm can achieve better results than in the case where the jobs were allocated based on their real size.

To test the effect of clustering on the LPT algorithm, we assumed that *the job run times are known at allocation time*, and that there is no error in this information.

As a control case for our simulations, we have verified that the LPT algorithm works better when there are relatively few possible job run times. Then we have conducted measurements to validate our hypothesis. The goal of these measurements was to see the effect of the number of possible job sizes on the performance and behaviour of the LPT algorithm; and deciding whether using K-means clustering can make as if there are less possible job sizes.

10.1 Measurement Parameters

We have defined the parameters of the measurements as shown in Table 2.

Deciding for a model distribution for job size is a difficult task, as real-world jobs are extremely unpredictable. However, the Pareto principle works in this case too i.e., a small number of jobs cause the majority

Table 2 Parameters applied in experiments

Parameter	Description	Range
Algorithm	The algorithm used to allocate jobs	LPT
$n \in \mathbb{N}$	Number of available resources	100
$k \in \mathbb{N}$	Number of jobs to be allocated	500
$t_i \in \mathbb{R}$	Run time of job $i \in [1..k]$	$\sim$ Gamma(μ, σ) where $\mu = 2$ and $\sigma = 500$ $\Longrightarrow E(t_i) = 1000$
c_i	Preload on DCI $i \in [1..n]$	{**0**, Convex, Concave, Linear} —See Fig. 1
K	Number of clusters in K-means clustering	[1..6] plus special case: unclustered

of the load on the computing infrastructure. Thus, we assume that any symbolic distribution that follows this principle may be an acceptable model for job size distribution. In our measurements, we used the Gamma distribution to generate job run-times. Gamma distribution was chosen based on experience: a series of extensive experiments showed the best match on job length distribution on our computing infrastructure, the SZTAKI Desktop Grid [26]. The focus on desktop grids is intentional: they are strongly related and especially taylored to embarrassingly parallel applications such as parameter sweeps or bag-of-tasks hence, they better reflect the conditions of a real execution environment.

The number of DCIs (n), the number of jobs to be allocated (k), the expected size of the jobs (μ), and the preload (c_i) are interdependent parameters that interfere with each other. We expect that if we fully explored their domains, we would find points in the parameter space where in each class the LPT algorithm would work similarly for each point.

Remark 5 For example, scaling the preload and the expected size of the jobs with the same constant would simply scale the turn-around time accordingly. Or, if we use twice as many DCIs with similarly distributed preload and twice as many jobs, the turn-around time would be the same. This is only a hypothesis, which we are yet to prove (or disprove) in the future. Nevertheless, the interference of these parameters is not in the scope of our measurements, as we are mainly interested in the behaviour of LPT regarding the number of possible job sizes. Therefore, we used fixed values for these parameters to provide a stable basis for our measurements.

For preload, we considered four cases: the special case when there is no preload at all, and three fixed preload vectors (Fig. 1), two of which have been generated offline from a Pareto distribution, which

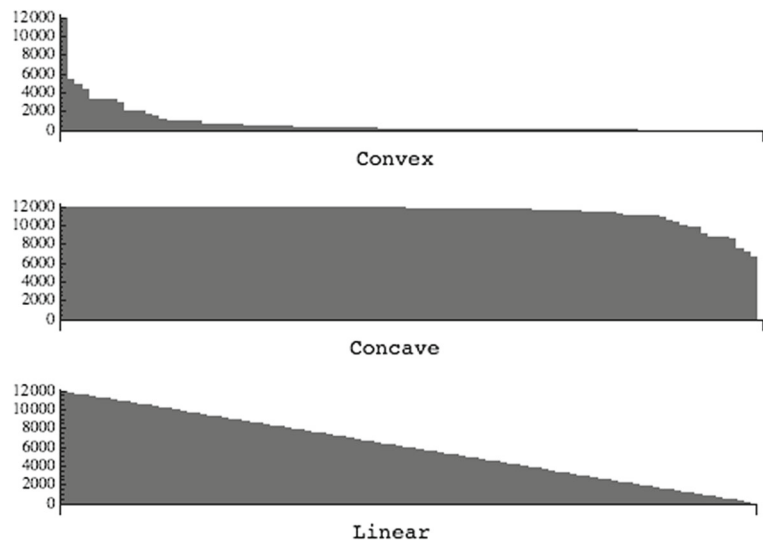

Fig. 1 Non-zero preloads (convex, concave, linear) used in the measurements

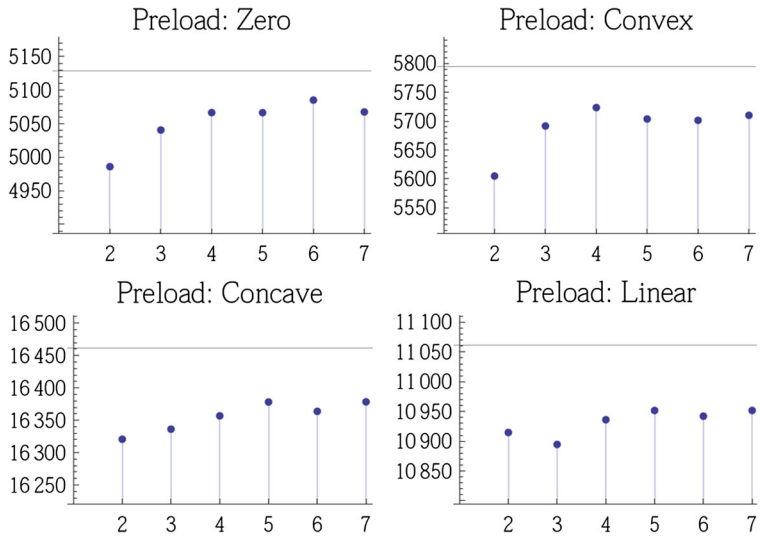

Fig. 2 Control-case results

captures real-life preloads well [2]. The difference between the three non-zero preloads is their shape: one is linear, one is convex, and one is concave. We will identify these preloads with these names in the discussion.

10.2 Measurement Method

Each parameter set described in Section 10.1 has been tested according to the following algorithm:

1. k jobs are generated with random run times t_i based on the Gamma distribution.
2. Each job is assigned to a representative element $\hat{t}_i$. This is done by either
   - clustering jobs into K distinct clusters, and assigning its cluster representative to each job; or
   - in the unclustered case, each job is assigned to itself as the representative.
3. The LPT algorithm is used to assign each job to a DCI, based on the representative associated with that job.
4. For each DCI, the local turn-around time is the sum of the *real* job run times (t_i), plus the preload. In the *control case*, the run time of the representative element ($\hat{t}_i$) is used as the job run time, making the number of possible job lengths equal to K.

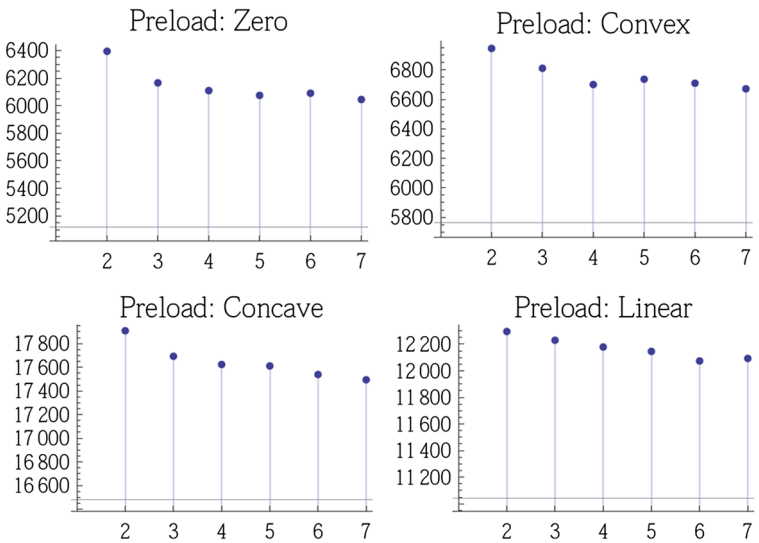

Fig. 3 Hypothesis validation results

5. The result of the measurement is the turn-around time, which is the maximum of the local turn-around times.
6. This is repeated 100 times for each parameter set to find average results.

10.3 Measurement results

We have performed 100 measurements for all parameter sets described in Section 10.1 to validate our hypothesis; and we repeated all measurements for the control case – resulting in a total of 2×16800 measurements. The averaged results of these measurements are shown in Figs. 2 and 3, as a function of K.

In each figure, the horizontal axis is the number of clusters (K), the vertical axis is the average turnaround time (smaller is better). In all cases, the horizontal line shows the *unclustered* result for that case.

The results of the control case is presented in Fig. 2. In this case, after allocating the jobs, we modified their run times to match that of their representative. That is, in each case, there was only K different possible values for job length. We can see that the LPT algorithm works better, indeed, when there are less possible job sizes. It is very clear from these figures, that the LPT algorithm performs considerably worse when job lengths can be of any value (the case represented by the horizontal lines). It can also be seen that as the number of possible job sizes grow, so do the turnaround times achieved by the LPT allocation.

However, as a consequence of our evaluations, our hypothesis proved to be wrong. In Fig. 3, we can see the case where the representative job sizes ($\hat{t}_i$) were used for allocation, but jobs were allowed to run for their real run time (t_i). The worst case is when all jobs are "clustered" into a single cluster; that is, while allocating them, all jobs are considered unit length, where the unit is the median of the job lengths. We can see that the turnaround time decreases as the number of clusters is increased, achieving the best turnaround time when the real job size, t_i is used to allocate jobs; that is, when we do not use clustering.

This is completely the opposite of the behaviour experienced in the control case. The unclustered allocation in the clustered case is actually equivalent to that in the control case; and indeed, it can be seen that the horizontal lines pertaining to the same preload closely match, as expected. For example, when there is no preload, both horizontal lines are between 5100 and 5150. As we increase K, the results of both cases tend towards this line; the control increases (worsens), while the clustered case decreases (improves) towards it.

In the next section we shall treat the possible answer to the questions involved.

11 Conclusions

In this work we addressed the resource allocation problem of parameter study jobs or bag-of-tasks applications, and aimed at systematically exploring the possible scenarios and techniques in a series of successively more complex allocation models for Distributed Computing Infrastructures. Table 3 summarizes the models and Fig. 4 shows the connections between them. The allocation problem is known to be NP hard even in the simplest cases and by revealing its constituting facets we examined this long studied problem

Table 3 The series of allocation models. In some cases, the notion in the column is not applicable or not known for the model – this is indicated in the corresponding entry

	Deterministic	Preload	Makespan optimisation	Complexity	Speciality
Model 1	yes	yes	yes	P	
Model 2	yes	yes	yes	P	$R_i = \beta_i k_i + \delta_i$
Model 3	yes	yes	yes	P	monotone, general
Model 4	no	yes	yes	P	
Model 5	no	no	yes	n/a	
Model 5'	no	no	yes	P	
Model 6	yes	yes/no	yes	NPH	

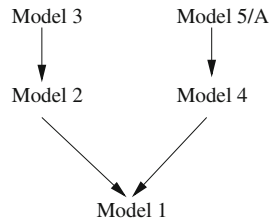

Fig. 4 The relationship between models. *Arrow* A → B means that model B is the special case of model A

area with novel insights. The outcome of our exploration is the series of models and their analysis with the aim to give hints for selecting the right approach at practical applications.

A model comprises certain elements and assumptions of the problem, these differ in the successive elements of the model series. Solutions to these problems are orthogonal: some problems have easy solutions, some others do not have any solution described by a polynomial algorithm. We did not provide solutions to each model in the series but demonstrated (i) an algorithm to a deterministic problem; (ii) reducing stochastic models to the simple deterministic model thus, solving a more complex problem based on a simple one; (iii) some methods and experiments to solve the more complex problems by heuristic means. In the latter case we provided an analysis of an approximate method based on clustering heuristics. In these measurements we focused on a few, synthetic cases of job-length and preload distributions. We have shown that using K-means clustering is not a viable option to imitate a small number of possible job-lengths.

There are several possible explanations for the negative result above. We mention here two of them. – The LPT algorithm yields much better results than the theoretically guaranteed ones, even for an enormous number of job sizes and thus it is not needed to decrease this number. – Keeping the frame of the method, a very special clustering technique is needed to yield the result expected in the original hypothesis. To find such a technique, is a challenge both in practical and theoretical sense. In the future, we shall work in this direction.

Acknowledgments The research leading to these results has received funding from the ER-Flow FP7 project under grant agreement 312579.

References

1. Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L., Myers, J.: Examining the challenges of scientific workflows. IEEE Comput. **40**(12), 26–34 (2007)
2. Bacsó, G., Visegrádi, A., Kertesz, A., Németh, Z.: On efficiency of multi-job grid allocation based on statistical trace data. J. Grid Comput. (2013)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
4. Lee, C.B., Schwartzman, Y., Hardy, J., Snavely, A.: Are User Runtime Estimates Inherently Inaccurate? vol. 3277, pp. 253–263. Springer LNCS (2005)
5. Ramírez-Alcaraz, J.M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., Gonzalez-García, J.L., Hirales-Carbajal, A.: Job allocation strategies with user run time estimates for online scheduling in hierarchical grids. J. Grid Comput. **9**(1), 95–116 (2011)
6. Schwiegelshohn, U., Tchernykh, A., Yahyapour, R.: Online scheduling in grids. In: 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS), vol. 2008, pp. 1–10 (2008)
7. Oprescu, A., Kielmann, T.: Bag-of-Tasks Scheduling under Budget Constraints. CloudCom, pp. 351–359 (2010)
8. Sgall, J.: On-line Scheduling - A Survey. Dagstuhl Seminar on On-Line Algorithms (Schloss Dagstuhl, Wadern, Germany, June 24-28, 1996), LNCS. Springer (1998)
9. Schuurman, P., Woeginger, G.: Approximation schemes – a tutorial. In: Möhring, R.H., Potts, C.N., Schulz, A.S., Woeginger, G.J., Wolsey, L.A. (eds.) Preliminary version of a chapter of the book "Lectures on Scheduling" (to appear)
10. Arndt, O., Freisleben, B., Kielmann, T., Thilo, F.: A comparative study of on-line scheduling algorithms for networks of workstation. Clust. Comput. **3**(2), 95–112 (2000)
11. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
12. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)
13. Xu, C., Lau, F.C.M.: Load Balancing in Parallel Computers: Theory and Practice. Springer Science & Business Media (1997)
14. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**(2), 416–429 (1969)
15. Smith, W.E.: Various optimizers for single-stage production. Nav. Res. Logist. Q. **3**, 5917 (1956)
16. Schrage, L.: A proof of the shortest remaining processing time processing discipline. Oper. Res. **16**, 687170 (1968)
17. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., Gonzalez-Garcia, J.L., Roblitz, T., Ramirez-Alcaraz, J.M.: Multiple workflow scheduling strategies with user run time estimates on a grid. J. Grid Comput. (2012)
18. Rényi, A.: Probability Theory. Elsevier (1970)
19. Silberstein, M., Sharov, A., Geiger, D., Schuster, A.: GridBot, execution of bags of tasks in multiple grids. In:

Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09) (2009)
20. Saha, D., Menasce, D., Porto, S.: Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. J. Parallel Distrib. Comput. **28.1**, 1–18 (1995)
21. Maheswaran, M., Ali, S., Siegal, H.J., Hensgen, D., Freund, R.F.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Proceedings of Heterogeneous Computing Workshop, 1999. (HCW'99), pp. 30-44. IEEE (1999)
22. Casanova, H., et al.: Heuristics for scheduling parameter sweep applications in grid environments. In: Proceedings of 9th Heterogeneous Computing Workshop (HCW 2000). IEEE (2000)
23. Cirne, W., Paranhos, D., Costa, L., Santos-Neto, E., Brasileiro, F., Sauve, J., Silva, F.A.B., Barros, C.O., Silveira, C.: Running bag-of-tasks applications on computational grids: The mygrid approach, pp. 407-416. IEEE (2003)
24. Da Silva, D.P., Cirne, W., Vilar Brasileiro, F.: Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. Euro-Par 2003 Parallel Processing, pp. 169–180. Springer Berlin Heidelberg (2003)
25. Nielsen, F.: Chernoff information of exponential families. CoRR, arXiv: 1102.2684 (2011)
26. SZTAKI Desktop Grid. http://szdg.lpds.sztaki.hu/szdg/ (2014)