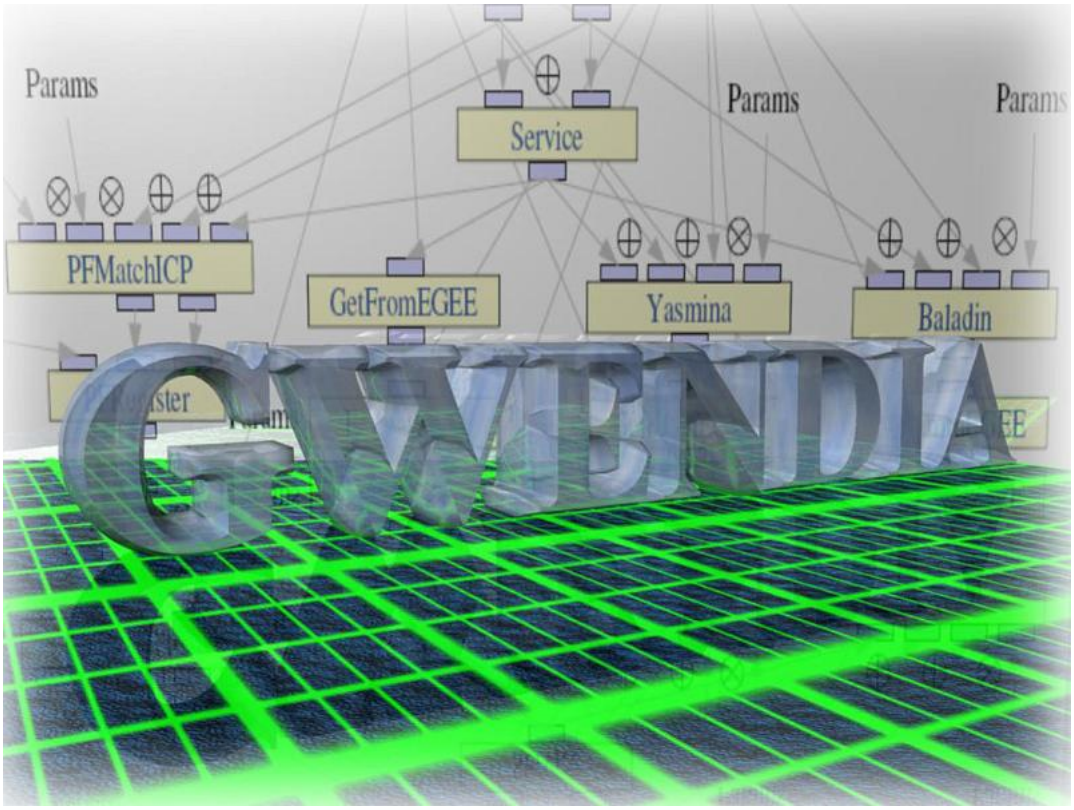
Gwendia ANR-06-MDCA-009
Grid Workflow Efficient Enactment for Data Intensive Applications

L1.2: Bibliography on Workflow Scheduling Heuristics



Raphaël BOLZE	GRAAL (LIP)	raphael.bolze@ens-lyon.fr
Frédéric DESPREZ	GRAAL (LIP)	frederic.desprez@ens-lyon.fr
Tristan Glatard	RAINBOW (I3S)	glatard@i3s.unice.fr
Johan Montagnat	RAINBOW (I3S)	johan@i3s.unice.fr

Abstract

We propose a survey on the different workflow scheduling heuristics on grid platforms. Our aim is to provide a state of art of the different techniques used to tackle the problem of scheduling workflow applications on distributed and heterogeneous platforms. We also take a close look at existing workflow management systems.

Contents

1 Introduction

Large problems coming from numerical simulation or life science can now solved through the Internet using grid middleware infrastructure. Several approaches co-exist to port applications on grid platforms like classical message-passing, batch processing, web portals, etc. Even if the first applications of grid platforms were made of large sets of independent tasks, real life applications consists of graphs of dependent tasks, also named workflows. The scheduling of these tasks over distributed, heterogeneous, and dynamic platforms is a big challenge for computer scientists.

Scheduling has attracted numerous researches from theory to pratical implementation in software environments. This tough problem [38] is even more complicated for real-life application workflows on non-dedicated environment. In this report, we survey the different techniques for scheduling application workflows on grids.

Several surveys are already available around scheduling for distributed systems. Among them the following reports are close to our researches: static algorithms [37], workflows scheduling [69], workflow management systems [67], and scheduling algorithms for Grid Computing [11].

The rest of this report is organized as follows. The first section (2) presents the modelization of the problem and the resources used. The second section presents a survey of scheduling heuristics for DAG on distributed platforms. The last section before the conclusion describes the algorithms taking into account QoS constraints (on data, budget, or deadlines) as well as algorithms chosen for existing workflow middleware.

2 Problem modelisation

2.1 DAG model

The DAG is a generic model of a workflow application consisting of a set of tasks (nodes) among which precedence constraints exist. It is represented by $G = (V, E)$, where V is the set of v tasks that can be executed on a subset of the available processors. E is the set of e directed arcs or edges between the tasks that maintain a partial order among them. The partial order introduces precedence constraints, i.e. if edge $e_{i,j} \in E$, then task v_j cannot start its execution before v_i completes. Matrix D of size $v \times v$ denotes the communication data size, where $d_{i,j}$ is the amount of data to be transferred from v_i to v_j . A task graph is a weighted graph. The weight w_i of a node v_i usually represents its computation cost. The weight of an edge stands for the communication requirement between the connected tasks (the amount of data that must be communicated between them). In a given task graph, a root node is called an entry task and a leaf node is called exit task. it is usually assumed that the task graph is a single-entry and single-exit one. If there is more than one exit or entry task, we can always connect them to a zero-cost pseudo exit or entry task with zero-cost edges. This will not affect the model.

Parallel scientific applications can be divided in two major classes: *data–* and *task–parallel* applications. The former consists in applying the same operation in parallel on different elements of a data set, while the latter is defined to be concurrent computations on different data sets. These two classes can be combined to get a simultaneous exploitation of data– and task–parallelism, so called *mixed–parallelism*. In mixed–parallel applications, several data–parallel computations can be executed concurrently in a task–parallel way. Mixed–parallelism programming employs a M–SPMD (Multiple SPMD) style which is

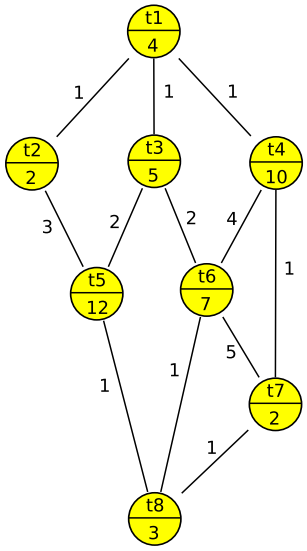


Figure 1: DAG example

the combination of both task–parallelism (MPMD) and data-parallelism (SPMD). Such an exploitation of mixed–parallelism has many advantages. One of them is the ability to increase scalability because it allows the use of more parallelism when the maximal amount of data– or task–parallelism that can be exploited is reached. A good overview of this topic is given in [4].

Most of the researches about the simultaneous exploitation of data– and task–parallelism have been done in the area of programming languages to give simple high level accesses to more parallelism and in the area of compilers, where problems such as scheduling and allocation of concurrent data–parallel tasks are studied [56].

2.2 Ressources model

There are a lot of possible way to define the resources model. In general, the resource model is modelized as a valued directed graph where vertices represent resources and edges represent connections between 2 resources. Let's define a set of resource R $\{r_u$, $1 \leq u \leq |R|\}$, and a set of connection C between 2 resources r_u and r_v, $c_{u,v}$. This defines the platform model $\wp = (R, C)$. The platform model is define by a topology : the shape of the graph. You can imagine a plethora of topology usual set of those used are : linear chain, bus , ring, strar, mesh, tree, hypercube, clique, and combinaison of the different regular topology.

Then, when the toplogy is chosen, the characteristic of the resources are defined : computing capacity, storage capacity ... , The connection are valued by a capacity or a bandwidth.

Classical topology for a computing platform is virtual clique of resources, each resource can communicate with each other. Then this model is decline into different variants :

1. fully homogeneous : all resources are homogeneous i.e they all have the same characteristic

2. fully uniform heterogeneous : all resources are heterogeneous, it exists a relation

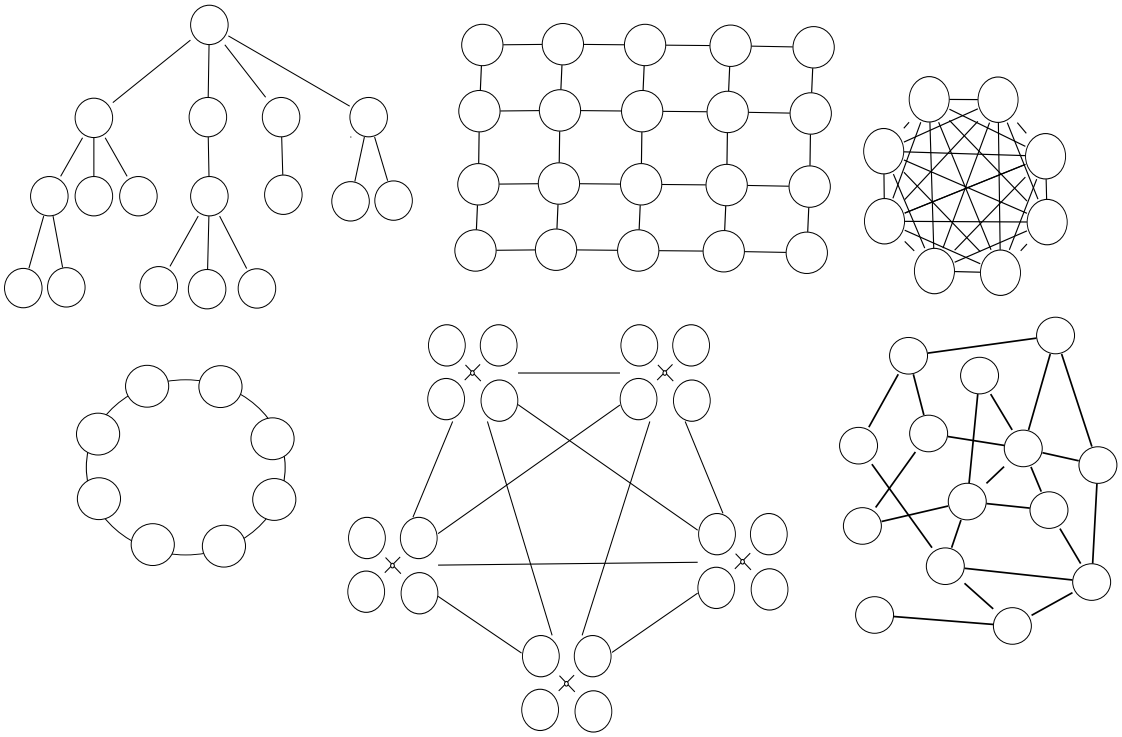


Figure 2: topology examples

between the properties of the resources i.e : one resources is twice faster than an other for all tasks

3. fully unrelated heterogeneous : all resources are heterogenous and it does not exist any relation between characteristics of the resources.

Some of research try to define a realistic communication model between resources, some others define a realistic model for the computation model. The difficulty in the workflow scheduling problem is that we have to take into acount 2 realistics models, one for the communication layer and one for the computation.

The choice of the resource model is important for the definition of the scheduling problem. All resource model try to fit a particular real world situation. Theoritical results usually depend on the resource model.

3 Scheduling heuristics

A plethora of heuristics have been proposed based on a wide spectrum of techniques, including branch-and-bound, integer-programming, searching, graph- theory, randomization, genetic algorithms, and evolutionary methods. The objective of this survey is to describe various scheduling algorithms and their functionalities in a contrasting fashion as well as examine their relative merits in terms of performance and time-complexity.

3.1 List scheduling

List scheduling heuristics maintain a list of nodes according to their priorities. A list scheduling algorithm repeatedly carries out the following steps:

(1) Take a task from a list of unscheduled ready task. A task becomes ready for assignment when all of its parents are scheduled.

(2) Select a suitable processor for assignment. Typically, a suitable processor is one that can execute the task the earliest.

(3) Assign the task the "suitable processor" and the task from the listof unschedule tasks.

The first step of the list scheduling is also known as *task prioritizing*, the second step as *processor selection*. We can distinguish static and dynamic list scheduling : the scheduling list is statically constructed before node allocation begins and the sequencing in the list is not modified. In some heuristics, the priority of unscheduled tasks are recomputed after each (or a certain number of) allocation then the scheduling list sequencing is rearranged. Those list scheduling heuristics are named as dynamic. Dynamic list scheduling can potentially generate better schedules. However, a dynamic approach can increase the time-complexity of the scheduling algorithm.

There are various ways to determine the priorities of nodes. Two frequently used attributes for assigning priority are the *t-level* (top level) and *b-level*(bottom level). The *t-level* of a task n_i is the length of a longest path (the longest path is not unique) from an entry node to n_i (excluding n_i). Here the length of a path is the sum of all the node and edge weights along the path. The *t-level* n_i highly correlates with n_is earliest start-time, denoted by $T_s(n_i)$, which is determined after n_i is scheduled to a processor. This is because after n_i is scheduled, its $T_s(n_i)$ is simply the length of the longest path reaching it. The *b-level* of a node n_i is the length of a longest path from n_i to an exit node. The *b-level* of a node is bounded from above by the length of a critical path. A critical path (CP) of a DAG, which is an important structure in the DAG, is a longest path in the DAG. Clearly, a DAG can have more than one CP. The *s-level* ... The time-complexity of a procedure to calculate the *t-level*, the *b-level* or the *s-level* is $O(e + v)$.

node	*b-level*	*t-level*	sl	ALAP
t_1	37	0	26	0
t_2	21	5	17	16
t_3	25	5	20	12
t_4	32	5	22	5
t_5	16	12	15	21
t_6	18	19	12	19
t_7	6	31	5	31
t_8	3	34	3	34

Different algorithms use the *t-level* and *b-level* in different ways. Some algorithms assign a higher priority to a node with a smaller *t-level* while some algorithms assign a higher priority to a node with a larger *b-level*. Still some algorithms assign a higher priority to a node with a larger (*b-level* - *t-level*). In general, scheduling in a descending order of *b-level* tends to schedule critical path nodes first, while scheduling in an ascending order

of *t-level* tends to schedule nodes in a topological order. The composite attribute (*b-level* - *t-level*) is a compromise between the previous two cases. If an algorithm uses a static attribute, such as *b-level* or *static b-level*, to order nodes for scheduling, it is called a static algorithm; otherwise, it is called a dynamic algorithm. The Tab 3.1 shows the value of the different attributes describe in the case of the DAG of the Figure 1.

	Comp. costs	Comm. cost	Priority	List type	Complexity
HLEFT	any	any	b-level	static	$O(v^2)$

List scheduling heuristics have been extensively used for mixed parallelism problems.

In [53], Ramaswamy introduces a structure to describe mixed–parallel programs: the Macro Dataflow Graph (MDG), a direct acyclic graph where nodes represent sequential or data–parallel computations and edges represent precedence constraints, with two distinguished nodes, one preceding and one succeeding all other nodes. Once the MDG is extracted from the code, a two step algorithm is applied to place and schedule tasks on the computing resources.

3.2 Clustering heuristics

Task clustering heuristic algorithms [39] have a number of properties in common when they try to achieve the goal of finding an optimal clustering for a DAG G. They all perform a sequence of clustering refinements starting with an initial clustering (initially each task is assumed to be in a cluster). Tasks are grouped into a set of clusters (unbounded) using linear or nonlinear clustering heuristics [16], [35], [66], [62] Each step performs a refinement of the previous clustering so that the final clustering satisfies or near to the original goals. The algorithms are non-backtracking, i.e., once the clusters are merged in a refinement step, they cannot be unmerged afterwards. A typical refinement step is to merge two clusters and *zero* the edge that connect them. Zeroing the communication cost on the edge between two clusters try to achieve the goal of reducing the makespan (or parallel time) of the schedule. in the second phase, clusters are mapped onto the set of available processors using communication sensitive or insensitive heuristics [41]. Complexity of clustering algorithms tends to be lower in comparison to list-based techniques because during the clustering phase, the former is not required to work on the limited processors constraint as is generally the case with the latter, and during the mapping phase, low complexity load balancing heuristics may be employed in clustering-based algorithms. However, comparison in terms of quality of schedule generated is left open [40].

3.3 Duplication based

from [1] Task duplication means scheduling a parallel program by redundantly allocating some of its tasks on which other tasks critically depend. This reduces the start times of waiting tasks which can eventually improve the overall execution time of the whole program. Duplication based scheduling can be particularly useful for systems with high communication overhead such as a network of workstations.

Duplication Scheduling Heuristic. The DSH (Duplication Scheduling Heuristic) algorithm [36] uses the static level (defined as the largest sum of computation costs along a path from the node to an exit node) as the priority for each node. The algorithm considers each node in descending order of their priorities. In examining the suitability of a

processor for a node, the algorithm first determines the start time of that node on the processor without duplication of any ancestor, and then considers the duplication time slot (the idle time period from the finish time of the last scheduled node on the processor and the start time of the node currently under consideration). If a suitable processor is found, the algorithm attempts to duplicate the parents of the node into the duplication time slot until either the slot is used up or the start time of the node does not improve further. This process is repeated for other processors and the node is scheduled to the processor that gives the smallest start time. The DSH algorithm calculates the priorities of nodes based on static levels which may not accurately capture the relative importance of nodes because dynamically changing communication costs (during the scheduling steps) among nodes are not taken into account. Furthermore, duplication may not always be very effective since the algorithm considers only one idle time slot on a processor.

PY algorithm. The PY algorithm (named after Papadimitriou and Yannakakis) [47] uses an attribute called the e-value to approximate the absolute achievable lower bound of the start time of a node. This attribute is computed recursively beginning from the entry nodes to the exit nodes. After computing the e-values, the algorithm inserts each node into a cluster in which a group of parents are duplicated such that the data arrival times from these ancestors are larger than the e-value of the node. It has been shown that the schedule length generated by the algorithm is within a factor of two from the optimal. The algorithm clusters nodes in a subgraph for duplication by using a node inclusion inequality which checks the message arrival times against the lower bound values of the candidate node under consideration. This can potentially leave out the nodes which are more important for reducing the start time of the given node, and this may lead to a poor schedule.

Lower Bound. We call the algorithm proposed in [9] the LWB (Lower Bound) algorithm based on its main procedure. The algorithm first determines the lower bound start time (denoted by lwb) for each node and then identifies a set of critical edges in the DAG. A critical edge is one in which a parents message available time for the child is greater than the lower bound start time of the child. Thus, the parent and child have to be scheduled to the same processor in order to reduce the start time of the child. Based on this idea, the LWB algorithm schedules every path of critical edges to a distinct processor. Since these paths may share ancestors, duplication is employed. It should be noted that the lwb value of a node is different from the e-value used in the PY algorithm in that the lwb value is computed by considering a single path from an entry node, while the e-value is computed by taking the whole subgraph reaching the node into account. The algorithm considers only those ancestors which are on a single path. When a node has more than one heavily communicated parent, this technique does not minimize the start time of the node (which can be done by duplicating more than one parents on a processor). Nevertheless, as is shown in [9], the LWB algorithm can generate optimal schedules for task graphs in which node weights are strictly larger than any edge weight.

Bottom-Up Top-Down Duplication Heuristic. The BTDH (Bottom-Up Top-Down Duplication Heuristic) algorithm [8] is essentially an extension of the DSH algorithm [36] described above. The major improvement brought by the BTDH algorithm over the DSH algorithm is that the former keeps on duplicating ancestors of a node even when the duplication time slot is filled up and the start time of the node under consideration

temporarily increases. This strategy is based on the intuition that the start time may eventually be reduced by duplicating all the necessary ancestors. As the BTDH algorithm also uses static level for priority assignment, it may not always accurately capture the relative importance of nodes.

Linear Clustering with Task Duplication. The LCTD (Linear Clustering with Task Duplication) algorithm [5] first iteratively clusters nodes into larger nodes. At each iteration, nodes on the longest path are clustered and removed from the task graph. This operation is repeated until all nodes in the graph are removed. After performing the clustering step, the LCTD algorithm identifies those edges among clusters that determine the overall completion time. The algorithm then attempts to duplicate the parents corresponding to these edges to reduce the start times of some nodes in the clusters. Linear clustering may not always accurately identify the nodes that should be scheduled to the same processor. In addition, in the context of duplication based scheduling, linear clustering prematurely constrains the number of processors used. This constraint can be detrimental because the start times of some critical nodes may possibly be significantly reduced by using a new processor in which its ancestors are duplicated.

Critical Path Fast Duplication. CPFD is an algorithm that based on partitioning the DAG into three categories: critical path nodes (CPN), in-branch nodes (IBN), and out-branch nodes (OBN). An IBN is a node from which there is a path reaching a CPN. An OBN is a node which is neither a CPN nor an IBN. Using this partitioning of the graph, the nodes can be or- dered in decreasing priority as a list called the CPN-Dominant Sequence. CPFD will then schedule all the nodes according to the CPN-Dominant Sequence following some rules.

Large Scale Optimal Algorithm. An optimal algorithm (LSOA) for task duplication based cluster scheduling. LSOA searches the whole problem space, therefore it promises optimal solution while not guarantee acceptable running time. LSOA has effective cutting strategies to reduce the search spaces. Although running time degenerates with the increment of CCR rapidly, LSOA can found optimal solutions using less time than sub-optimal algorithms such as CPFD in many large scale cases

Task duplication-based scheduling Algorithm for Network of Heterogeneous systems. Task duplication-based scheduling Algorithm for Network of Heterogeneous systems (TANH) [3], [54],[55]

3.4 Metaheuristics

from [43] The family of metaheuristics includes, but is not limited to, adaptive memory procedures, ant systems, evolutionary methods, mimetic algorithms, variable neighborhood search, greedy randomized adaptive search, scatter search, neural networks, and their hybrids. For an extensive survey on metaheuristic strategies see [34, 57]. Three strategies have been most popular and successful from among the metaheuristics over the years: simulated annealing, tabu search, and genetic algorithms.

Metaheuristics are a class of approximate methods that have been developed dramatically since their inception in the early 1980s. They are designed to attack com- plex optimization problems where classical heuristics and optimization methods have failed to

be effective and efficient. A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions Local (neighbourhood) search methods form a general class of heuristics to obtain near-optimal (approximate) solution with a reasonable computational time. For a minimisation problem, the local search descent method is the simplest neighbour- hood search algorithm. It starts from an initial solution as its current solution. It then explores the vicinity of this solution using a certain mechanism to generate neigh- bouring solutions. Neighbours are then accepted to replace the current solution if they improve upon it. The search continues until no further improvement can be made and the algorithm terminates with a local optimum. The local optimum produced by the local search procedure can be very far from optimality. In order to avoid such disadvantages while maintaining the simplicity and generality of the approach, the following concepts which form the basis of most metaheuristics are normally consid- ered:

(1) Start from good initial solutions which can be generated intelligently using greedy random adaptive search procedures [14] or space-search methods [61]

(2) Use the learning strategies of neural networks [60] and tabu search [25] that gather information during the algorithms execution in order to guide the search to find possibly better solutions.

(3) Employ non-monotonic search strategies that sample accept neighbours based on hybrid modifications of simulated annealing and tabu search methods or others [23, 24]; [63]; [44, 46, 45], and [29].

(4) Introduce a more complex neighbourhood mechanism, such as ejection chains, see [19]; [12] and [6] and compound moves with proper data structures, [22]; [44] and [30].

(5) Employ a solution-generation mechanism that works on a set of solutions rather than a single solution, as in genetic algorithms [42], evolutionary programming [15] and scatter search [21].

3.4.1 Constraint Logic Programming

Constraint Logic Programming (CLP) is a field of research in Artificial Intel- ligence [64]. CLP techniques have recently attracted much interest and become commercially adopted due to their ability to represent and solve constraint satisfaction problems, and are thus suitable for dealing with scheduling problems [31]. In CLP, a problem is formulated in terms of finite domain variables and the constraints imposed upon them. Finite domain variables are those which can be assigned a finite number of integer values. A solution is found through enumeration of a tree search. A CLP technique would generate values for the variables, propagating values through the constraints in order to prune parts of the solution space where inconsistencies are discovered. At each step of the enumeration, consistencies between the original constraints and those added during the process are kept by altering the domains of the relevant variables through removing certain integer values from their domains. Constraints can force a variable to have an empty domain, indicating no possible value to satisfy all the constraints. Therefore, the added con- straint

is violated; branching is then halted and backtracking takes place. The basic method is therefore a backtrack search with a look-ahead mecha- nism. Look-ahead algorithms propagate constraints after assigning a value to each variable in order to reduce the search space and anticipate dead-ends (Haralick and Eliott, 1980). A CLP method does not need to be programmed by the user, but is provided by the CLP languages, such as PROLOG. The price for such flexibility is a loss of efficiency in solving even a moderate size problem. However, intelligent back- tracking, learning and other hybrid algorithms are often used to improve the CLP performance [17]; [50]; [10] and [51].

3.4.2 Natural evolutionary computation

The field of natural evolution is in a stage of tremendous growth. The most obvious approach is to characterize the field in terms of its historical evolution. There are currently three well-defined paradigms which have served as the basis for much of the research in the field: Genetic Algorithms (**GA**); Evolution Strategies (**ES**); Evolutionary Programming (). Each of these paradigms emphasizes a different facet of natural evolution. In general, these paradigms are based on a population-based optimization process. Simulating this process on a computer results in stochastic optimization techniques that can often outperform classical methods of optimization when applied to difficult real-world problems. Historically, there are associations between GAs and binary string representations of solutions, between ESs and vectors of real numbers and between EPs and finite state machines. The EP and ES communities have emphasized a mechanism for reproduction based on mutation. By contrast, the GA community emphasized reproduction based on recombination and mutation.

Genetic algorithms. Genetic algorithms (GAs) originated from the studies of cellular automata conducted by John Holland and his colleagues. Hollands book in 1975 is generally acknowledged as the beginning of GA research and only recently has their potential for solving combinatorial optimization problems been explored. Genetic algorithms owe their name to an early emphasis on representing and manipulating individuals in terms of their genetic make-up rather than using a phenotypic representation. The basic idea is to maintain a population of candidate solutions which evolves under a selective pressure that favours the survival of the fittest. Hence they can be viewed as a class of local search methods that employ a solution-generation mechanism which operates on attributes of a set of solutions rather than attributes of a single solution using a move-generation mechanism.

The fundamental idea of GA is that it operates on a finite population of N chromosomes (solutions). The chromosomes are fixed strings with binary values (alleles) at each position (or locus). An allele is the 0 or 1 value in the bit string, and the locus is the position at which the 0 or 1 value is present in each location of the chromosome. Each chromosome of the population is evaluated according to some fitness function. Members of the population are selectively interbred in pairs to produce offspring. The fitter a member of the population the more likely it is to produce offspring. Genetic operators are used to facilitate the breeding process that results in offspring inheriting properties from their parents. The offspring are evaluated and placed in the population, possibly replacing the weaker members of the last generation. Thus, the search mechanism consists of three phases: evaluation of the fitness of each chromosome, selection of the parent chromosomes, and applications of mutation and re-combination (crossover) operators to the parent chromosomes. The new chromosomes resulting from these operations form the population for

the next generation and the process is repeated until the system ceases to improve. The survival of the fittest principle ensures that the overall quality of solutions increases as the algorithm progresses from one generation to the next.

3.4.3 Neural networks

Artificial neural networks are very powerful in scientific and engineering applications when used to predict, classify or recognize patterns due to the inherent data classification capabilities and massively parallel processing power. They have not been as successful when applied to optimization problems and are not competitive with the best metaheuristics from the operations research literature, when applied to combinatorial optimization problems. The interest in using neural networks for combinatorial optimization problems began in the mid-1980s with the work of [32, 33] on the travelling salesman problem. There are two basic approaches to using neural networks for solving optimization problems. The first approach is based on statistical physics and includes the Hopfield- Tank and the Mean-Field annealing neural networks algorithms. In the Hopfield-Tank approach, the set of all possible solutions of a combinatorial optimization problem is mapped onto the set of all possible states of the network. The network is normally composed of one-layered neurons with fully weighted connections. An energy function is defined over the set of all possible states and it depends on the weights of the connections of the network. The neural networks algorithm iterates towards a ground state of minimum energy that corresponds to an optimal solution satisfying all constraints of the combinatorial optimization problem. Due to the feedback characteristics, the Hopfield-Tank model converges to a local minimum. Various modifications have been proposed to alleviate this problem such as the incorporations of simulated annealing in a Boltzmann machine and other stochastic models. The second approach is based on competitive neural networks and includes the Kohonen self-organizing network and the deformable (elastic) nets in that neurons are allowed to compete to become active under certain conditions. The aim of the Kohonen model is to find a mapping from a high dimensional input space onto a lower dimensional discrete lattice (usually two) of formal neurons. The mapping tries to reflect the topological neighbourhood relationships among the inputs in the arrangement of the corresponding neurons in the lattice. The above two approaches suggest neural networks as an alternative for solving certain optimization problems as compared to classical optimization techniques and other novel approaches.

3.4.4 simulated annealing

The simulated annealing (**SA**) metaheuristic has its origins in statistical physics. The interest in using SA for combinatorial optimization problems began in the early 1980s with the work of [58] and [7]. The SA metaheuristic performs a stochastic search of the neighbourhood space. Modifications to the current solution that increase the value of the objective function are allowed (for a minimisation problem) as opposed to classical descent methods where only modifications that decrease the objective value are allowed. More precisely, a modification that reduces the objective value is always accepted, while a modification that increase the objective value by Δ is accepted with a probability $\exp(\frac{\Delta}{T})$, where T is the temperature control parameter. At a high temperature, the probability of accepting an increase to the objective value is high. This probability gets lower as the temperature is decreased. Typically, the SA algorithm starts with a high temperature to perform a coarse search of the solution space. The temperature is then gradually reduced

to focus on a specific region. The initial temperature value of T, the number of iterations to be performed at each temperature, the cooling (reduction) rate of the temperature value and the stopping criterion are determined by the so-called SA cooling schedule.

3.4.5 Tabu search

Tabu search (**TS**) ideas were introduced by [18] and [28]. The TS method is a metaheuristic which shares with the SA algorithm the ability to guide the local search descent method to avoid bad local optima. It uses, however, a deterministic rather than stochastic acceptance criterion. At each iteration, TS moves to the best admissible neighbour, even if this causes the objective function to deteriorate. This admission criterion may lead to cycling, i.e., returning to solutions already visited. To avoid cycling, attributes of accepted solutions are stored in a tabu list and declared tabu for a number of iterations. A neighbouring solution is considered forbidden and deemed not admissible if it has attributes on the tabu list. Storing attributes rather than the complete solutions may cause some non-tabu solutions to be wrongly prevented and aspiration criteria are normally used to correct such errors. To produce good results, intensification and diversification strategies are also employed. The former strategy is used to perform a thorough search in good regions while the latter strategy is attempted to consider solutions in a broad area of the search. For more recent developments, refer to [23, 20].

4 Scheduling Workflows

Even if workflows can be modeled as DAGs, their scheduling on actual Grid infrastructure relies on the coordination of different heuristics taking into account its heterogeneity, its dynamicity and the data management. In this section, we survey papers dealing with the scheduling of workflows on Grids and the way they are implemented in given workflow middlewares.

4.1 Scheduling with Constraints

Many workflow applications require some quality-of-service assurance depending wheter they need to be executed before a specific time, cost less than a given amount of budget or use less than a given amount of MBytes. In [69], several such algorithms are described.

4.1.1 Storage constraints

[52]

4.1.2 Budget constraints

In [68], ...
In [59], ...

4.1.3 Deadline constraints

[48]

4.2 Scheduling strategies of specific Workflow systems

4.2.1 Pegasus

Pegasus [49] uses workflow partionning.

4.2.2 Askalon

The Askalon system [2, 13] is built upon a set of components for the development and execution of workflow application over the Grid. The *Resource Manager* is responsible for negociation, reservation, and allocation of resource, as well as automatic deployment of services. The Scheduler is a service that determines effective mappings of single and multiple workflow applications over the Grid using graph based heuristics and optimization alogorithms

The Askalon system uses three algorithms [65] to schedule workflows represented as DAGs (written in the AGWL language): HEFT, a genetic algorithm and a Myopic algorithm. This last algorithm is just a just-in-time scheduling strategy using local decisions. The conclusions of the authors is that HEFT performs better than the Myopic algorithm. The genetic algorithm can be computationally extensive. An other conclusion is that incremental workflow scheduling does not give the best performance and moreover, it tends to produce less efficient results for unbalanced workflows. Full-graph scheduling produces the best results.

4.2.3 Gridflow

Gridflow [27]

4.2.4 GrADS

GrADS [26]

5 Conclusion

References

[1] Ishfaq Ahmad and Yu-Kwong Kwok. A new approach to scheduling parallel programs using task duplication. In *ICPP*, pages 47–51, 1994.

[2] ASKALON. http://www.askalon.org/.

[3] Rashmi Bajaj and Dharma P. Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE Trans. Parallel Distrib. Syst.*, 15(2):107–118, 2004.

[4] Henri Bal and Matthew Haines. Approaches for Integrating Task and Data Parallelism. *IEEE Concurrency*, 6(3):74–84, Jul-Sep 1998.

[5] Jeff Marquis Behrooz Shirazi, Hsing-Bung Chen. Comparative study of task duplication static scheduling versus clustering and non-clustering techniques. *Concurrency: Practice and Experience*, 7(5):371–389, 1995.

[6] C. Roucairol C. Rego. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. *Metaheuristics. Theory and Applications*, 1996.

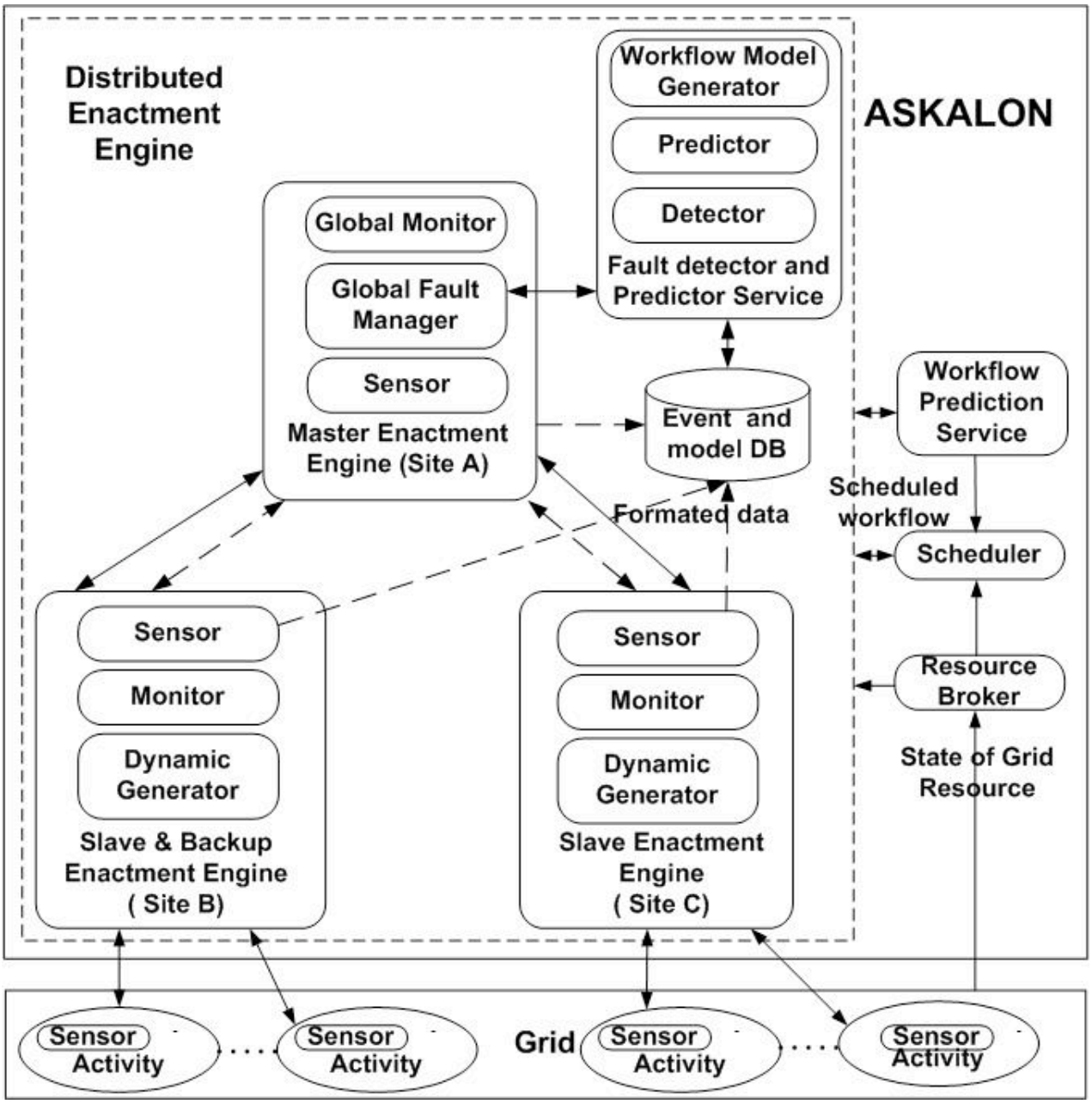


Figure 3: Askalon scheduling architecture

[7] V. Cerny. A thermodynamical approach to the travelling salesman problem. an efficient simulated annealing algorithm. *Journal of Optimization Theory and Applications*, 45:41, 1985.

[8] Y.-C. Chung and S. Ranka. Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 512–521, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[9] J-Y. Colin and P. Chrtienne. C.p.m. scheduling with small communication delays and task duplication. *Operations Research*, 39(3):680–684, 1991.

[10] R. Dechter D. Fost. Dead-end driven learning. In *Proceedings of the 12th National Conference for Artificial Intelligence (AAAI94)*, page 294, 1994.

[11] Fangpeng Dong and Selim G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 2006-504, Queen's University, School of Computing, January 2006.

[12] I.H. Osman F. Glover, E. Pesch. Efficient facility layout planning. *technical report*, 1995.

[13] T. Fahringer, R. Prodan, Rubing Duan, F. Nerieri, S. Podlipnig, Jun Qin, M. Siddiqui, Hong-Linh Truong, A. Villazon, and M. Wieczorek. Askalon: A grid application development and computing environment. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 122–131, Washington, DC, USA, 2005. IEEE Computer Society.

[14] T. Feo and M. Resende. Greedy randomized adaptive search procedures, 1995.

[15] D.B. Fogel. A comparison of evolutionary programming and genetic algorithms on selected constrained optimization problems. *Simulation*, 64:397, 1995.

[16] A. Gerasoulis and T. Yang. A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs onto Multiprocessors. *Parallel and Distributed Computing,*, 16(4):276–291, 1992.

[17] R.S. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25, 1993.

[18] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533, 1986.

[19] F. Glover. New ejection chain and alternating path methods for traveling salesman problems. *Operations Research and Computer Science. New Developments in Their Interfaces*, 1992.

[20] F. Glover. Tabu search. improved solution alternatives. *Mathematical Programming State of the Art*, 1994.

[21] F. Glover. Scatter search and star-paths. beyond the genetic metaphor. *OR Spektrum*, 17:125, 1995.

[22] F. Glover. Multilevel tabu search and embedded search neighbourhoods for the travelling salesman problem. *ORSA Journal on Computing*, 1996.

[23] Fred Glover. Tabu search fundamentals and uses. *working paper*, 1995.

[24] Fred Glover. Tabu thresholding. improved search by non-monotonic trajectories. *ORSA journal on computing*, 7:426, 1995.

[25] Fred Glover and Fred Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[26] GrADS. http://www.hipersoft.rice.edu/grads/.

[27] Gridflow. http://gridflow.ca/.

[28] P. Hansen. Future paths for integer programming and links to artificial intelligence. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, 1986.

[29] N. Christofides I.H. Osman. Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operational Research*, 1:317, 1994.

[30] S. Salhi I.H. Osman. Local search strategies for the mix fleet vehicle routing problem. *Local search strategies for the mix fleet vehicle routing problem*, 1996.

[31] K. Darby-Dowman J. Little. The significance of constraint logic programming to operational research. *Operational Research Tutorial Papers*, 1995.

[32] Tank J.J. Hopfield, D. Neural computations of decisions in optimization problems. *Biological Cybernetics*, 52:141, 1985.

[33] Tank J.J. Hopfield, D. Computing with neural circuits. a model. *Science*, 233:624, 1986.

[34] James P. Kelly. *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[35] S.J. Kim and J.C. Browne. A general approach to mapping of parallel computation upon multiprocessor architectures. In *Int'l Conf. Parallel Processing*, volume 2, pages 1–8, 1988.

[36] Boontee Kruatrachue and Ted Lewis. Grain size determination for parallel processing. *IEEE Softw.*, 5(1):23–32, 1988.

[37] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, 1999.

[38] Joseph Y-T Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC, 2004.

[39] J. Liou and M. Palis. An efficient task clustering heuristic for scheduling dags on multiprocessors, 1996.

[40] D.S.L. Wei M.A. Palis, J.-C. Liou. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Trans. Parallel and Distributed Systems*, 7(1):46–55, 1996.

[41] K. Steiglitz M.D. Dikaiakos, A. Rogers. A comparative study of heuristics for mapping parallel algorithms to message passing multiprocessors. *technical report*, 1994.

[42] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, 1996.

[43] Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz, editors. *Grid resource management: state of the art and future trends*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[44] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451, 1993.

[45] I.H. Osman. Heuristics for the generalized assignment problem. simulated annealing and tabu search approaches. *OR Spektrum*, 17:211, 1995.

[46] I.H. Osman. An introduction to metaheuristics. *Operational Research Tutorial paper*, 1995.

[47] Christos H. Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, 19(2):322–328, 1990.

[48] Y. Patel, A.S. McGough, and J. Darlington. QoS Support For Workflows In a Volatile Grid. In *Grid Computing Conference*, pages 64–71, 2006.

[49] PEGASUS. http://pegasus.isi.edu/.

[50] P. Posser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268, 1993.

[51] J. Pearl R. Dechter. Network based heursitics for the constraint satisfaction problems. *Artifical Intelligence*, 34:1, 1988.

[52] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi. Scheduling Data-Intensive Workflows onto Storage-Constrained Distributed Resources. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID'07)*. IEEE, 2007.

[53] S. Ramaswany. *Simultaneous Exploitation of Task and Data Parallelism in Regular Scientific Applications*. PhD thesis, University of Illinois at Urbana-Champaign, 1996.

[54] Samantha Ranaweera and Dharma P. Agrawal. A scalable task duplication based scheduling algorithm for heterogeneous systems. In *ICPP '00: Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing*, page 383, Washington, DC, USA, 2000. IEEE Computer Society.

[55] Samantha Ranaweera and Dharma P. Agrawal. A task duplication based scheduling algorithm for heterogeneous systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico, May 2000.

[56] T. Rauber and G. Rnger. Scheduling of data parallel modules for scientific computing. In SIAM, editor, *Proceedings of Ninth SIAM Conference on Parallel Processing for Scientific Computing (PP99)*, San Antonio, Texas, March 1999.

[57] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, New York, NY., 1995.

[58] P.M. Vecchi S. Kirkpatrick, C.D. Gelatt. Optimization by simulated annealing. *Science*, 220:671, 1983.

[59] Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, and Marios Dikaiakos. Scheduling workflows with budget constraints. *Integrated research in Grid Computing, Gorlatch Sergei, Danelutto Morco (Eds.)*, 2006.

[60] R. Sharda. Neural networdl for the ms/or analyst. an application bibliography. *Interfaces*, 24:116, 1994.

[61] Robert H. Storer, S. David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Manage. Sci.*, 38(10):1495–1509, 1992.

[62] A. Gerasoulis T. Yang. Dsc: Scheduling parallel tasks on an unbounded number of processors. *EEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, 1994.

[63] C.W.A. Tsao T.C. Hu, A.B. Khang. A new class of non-monotonic threshold accepting methods. *ORSA journal on computing*, 7:417, 1995.

[64] EP.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, 1993.

[65] Marek Wieczorek, Radu Prodan, and Thomas Fahringer. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Rec.*, 34(3):56–62, 2005.

[66] Tao Yang. *Scheduling and code generation for parallel architectures*. PhD thesis, Rutgers University, New Brunswick, NJ, USA, 1993.

[67] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing, 2005.

[68] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3-4):217–230, 2006.

[69] Jia Yu and Rajkumar Buyya. Workflow scheduling algorithms for grid computing. Technical Report GRIDS-TR-2007-10, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, May 2007.