Technical Report No. 2006-504

Scheduling Algorithms for Grid Computing: State of the Art and Open Problems

Fangpeng Dong and Selim G. Akl

School of Computing,
Queen's University
Kingston, Ontario
January 2006

Abstract:

Thanks to advances in wide-area network technologies and the low cost of computing resources, Grid computing came into being and is currently an active research area. One motivation of Grid computing is to aggregate the power of widely distributed resources, and provide non-trivial services to users. To achieve this goal, an efficient Grid scheduling system is an essential part of the Grid. Rather than covering the whole Grid scheduling area, this survey provides a review of the subject mainly from the perspective of scheduling algorithms. In this review, the challenges for Grid scheduling are identified. First, the architecture of components involved in scheduling is briefly introduced to provide an intuitive image of the Grid scheduling process. Then various Grid scheduling algorithms are discussed from different points of view, such as static vs. dynamic policies, objective functions, applications models, adaptation, QoS constraints, strategies dealing with dynamic behavior of resources, and so on. Based on a comprehensive understanding of the challenges and the state of the art of current research, some general issues worthy of further exploration are proposed.

1. Introduction

The popularity of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components are changing the way we use computers today. These technical opportunities have led to the possibility of using geographically distributed and multi-owner resources to solve large-scale problems in science, engineering, and commerce. Recent research on these topics has led to the emergence of a new paradigm known as *Grid computing* [19].

To achieve the promising potentials of tremendous distributed resources, effective and efficient scheduling algorithms are fundamentally important. Unfortunately, scheduling algorithms in traditional parallel and distributed systems, which usually run on homogeneous and dedicated resources, e.g. computer clusters, cannot work well in the new circumstances [12]. In this paper, the state of current research on scheduling algorithms for the new generation of computational environments will be surveyed and open problems will be discussed.

The remainder of this paper is organized as follows. An overview of the Grid scheduling problem is presented in Section 2 with a generalized scheduling architecture. In Section 3, the progress made to date in the design and analysis of scheduling algorithms for Grid computing is reviewed. A summary and some research opportunities are offered in Section 4.

2. Overview of the Grid Scheduling Problem

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [45]. It is a shared environment implemented via the deployment of a persistent, standards-based service infrastructure that supports the creation of, and resource sharing within, distributed communities. Resources can be computers, storage space, instruments, software applications, and data, all connected through the Internet and a middleware software layer that provides basic services for security, monitoring, resource management, and so forth. Resources owned by various administrative organizations are shared under locally defined policies that specify what is shared, who is allowed to access what, and under what conditions [48]. The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [44].

From the point of view of scheduling systems, a higher level abstraction for the Grid can be applied by ignoring some infrastructure components such as authentication, authorization, resource discovery and access control. Thus, in this paper, the following definition for the term *Grid* adopted: "*A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements*" [10].

To facilitate the discussion, the following frequently used terms are defined:

- A ***task*** is an atomic unit to be scheduled by the scheduler and assigned to a resource.
- The ***properties*** of a task are parameters like CPU/memory requirement, deadline, priority, etc.
- A ***job*** (or ***metatask***, or ***application***) is a set of atomic tasks that will be carried out on a set of resources. Jobs can have a recursive structure, meaning that jobs are composed of sub-jobs and/or tasks, and sub-jobs can themselves be decomposed further into atomic tasks. In this paper, the term *job*, *application* and *metatask* are interchangeable.
- A ***resource*** is something that is required to carry out an operation, for example: a processor for data processing, a data storage device, or a network link for data transporting.
- A ***site*** (or ***node***) is an autonomous entity composed of one or multiple resources.
- A ***task scheduling*** is the mapping of tasks to a selected group of resources which may be distributed in multiple administrative domains.

2.1 The Grid Scheduling Process and Components

A Grid is a system of high diversity, which is rendered by various applications, middleware components, and resources. But from the point of view of functionality, we can still find a logical architecture of the task scheduling subsystem in Grid. For example, Zhu [123] proposes a common Grid scheduling architecture. We can also generalize a scheduling process in the Grid into three stages: resource discovering and filtering, resource selecting and scheduling according to certain objectives, and job submission [94]. As a study of scheduling algorithms is our primary concern here, we focus on the second step. Based on these observations, Fig. 1 depicts a model of Grid scheduling systems in which functional components are connected by two types of data flow: resource or application information flow and task or task scheduling command flow.

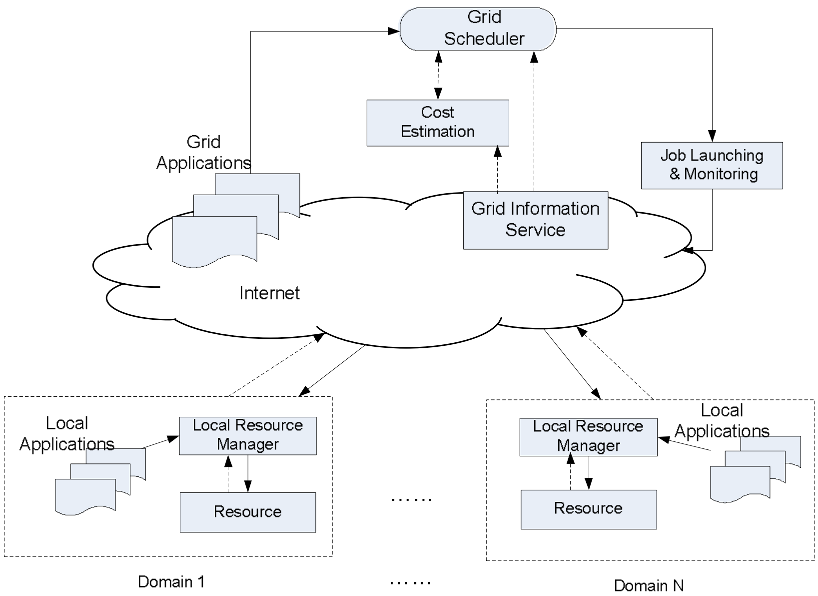

Fig. 1: A logical Grid scheduling architecture: broken lines show resource or application information flows and real lines show task or task scheduling command flows.

Basically, a Grid scheduler (GS) receives applications from Grid users, selects feasible resources for these applications according to acquired information from the Grid Information Service module, and finally generates application-to-resource mappings, based on certain objective functions and predicted resource performance. Unlike their counterparts in traditional parallel and distributed systems, Grid schedulers usually cannot control Grid resources directly, but work like brokers or agents[13], or even tightly coupled with the applications as the application-level scheduling scheme proposes [11], [105]. They are not necessarily located in the same domain with the resources which are visible to them. Fig. 1 only shows one Grid scheduler, but in reality multiple such schedulers might be deployed, and organized to form different structures (centralized, hierarchical and decentralized [55]) according to different concerns, such as performance or scalability. Although a Grid level scheduler (or *Metascheduler* as it is sometime referred to in the literature, e.g., in [77]) is not an indispensable component in the Grid infrastructure (e.g., it is not included in the Globus Toolkit [125], the defacto standard in the Grid computing community), there is no doubt that such a scheduling component is crucial for harnessing the potential of Grids as they are expanding quickly, incorporating resources from supercomputers to desktops. Our discussion on scheduling algorithms is based on the assumption that there are such schedulers in a Grid.

Information about the status of available resources is very important for a Grid scheduler to make a proper schedule, especially when the heterogeneous and dynamic nature of the Grid is taken into account. The role of the Grid information service (GIS) is to provide such information to Grid schedulers. GIS is responsible for collecting and predicting the resource state information, such as CPU capacities, memory size, network bandwidth, software availabilities and load of a site in a particular period. GIS can answer queries for resource information or push information to subscribers. The Globus Monitoring and Discovery System (MDS) [33] is an example of GIS.

Besides raw resource information from GIS, application properties (e.g., approximate instruction quantity, memory and storage requirements, subtask dependency in a job and communication volumes) and performance of a resource for different application species are also necessary for making a feasible schedule. Application profiling (AP) is used to extract properties of applications, while analogical benchmarking (AB) provides a measure of how well a resource can perform a given type of job [67] [96]. On the basis of knowledge from AP and AB, and following a certain *performance model* [14], cost estimation computes the cost of candidate schedules, from which the scheduler chooses those that can optimize the objective functions.

The Launching and Monitoring (LM) module (also known as the "binder" [31]) implements a finally-determined schedule by submitting applications to selected resources, staging input data and executables if necessary, and monitoring the execution of the applications. A good example of an LM is the Globus GRAM (Grid Resource Allocation and Management) [32].

A Local Resource Manager (LRM) is mainly responsible for two jobs: local scheduling inside a resource domain, where not only jobs from exterior Grid users, but also jobs from the domain's local users are executed, and reporting resource information to GIS. Within a domain, one or multiple local schedulers run with locally specified resource management policies. Examples of such local schedulers include OpenPBS [126] and Condor [127]. An LRM also collects local resource information by tools such as Network Weather Service [111], Hawkeye [127] and Ganglia [89], and report the resource status information to GIS.

2.2 Challenges of Scheduling Algorithms in Grid Computing

Scheduling algorithms have been intensively studied as a basic problem in traditional parallel and distributed systems, such as symmetric multiple processor machines (SMP), massively parallel processors computers (MPP) and cluster of workstations (COW). Looking back at such efforts, we find that scheduling algorithms are evolving with the architecture of parallel and distributed systems. Table 1 captures some important features of parallel and distributed systems and typical scheduling algorithms they adopt.

Table 1: Evolution of scheduling algorithms with parallel and distributed computing systems

Typical Architecture	DSM, MPP	COW	Grid
Chronology	Late 1970s	Late 1980s	Mid 1990s
Typical System Interconnect	Bus , Switch	Commercial LAN, ATM	WAN/Internet
Cost of Interconnection	Very Low/ Negligible	Low/ Usually Not Negligible	High / Not Negligible
Interconnection Heterogeneity	None	Low	High
Node Heterogeneity	None	Low	High
Single System Image	Yes	Yes	No
Resource Pool Static/Dynamicity	Predetermined and Static	Predetermined and Static	Not Predetermined and Dynamic
Resource Management Policy	Monotone	Monotone	Diverse
Typical Scheduling Algorithms	Homogeneous Scheduling Algorithms	Heterogeneous Scheduling Algorithms	Grid Scheduling Algorithms

Although we can look for inspirations in previous research, traditional scheduling models generally produce poor Grid schedules in practice. The reason can be found by going through the assumptions underlying traditional systems [14]:

- All resources reside within a single administrative domain.
- To provide a single system image, the scheduler controls all of the resources.
- The resource pool is invariant.
- Contention caused by incoming applications can be managed by the scheduler according to some policies, so that its impact on the performance that the site can provide to each application can be well predicted.
- Computations and their data reside in the same site or data staging is a highly predictable process, usually from a predetermined source to a predetermined destination, which can be viewed as a constant overhead.

Unfortunately, all these assumptions do not hold in Grid circumstances. In Grid computing, many unique characteristics make the design of scheduling algorithms more challenging [123], as explained in what follows.

- **Heterogeneity and Autonomy**

Although heterogeneity is not new to scheduling algorithms even before the emergence of Grid computing, it is still far from fully addressed and a big challenge for scheduling algorithm design and analysis. In Grid computing, because resources are distributed in multiple domains in the Internet, not only the computational and storage nodes but also the underlying networks connecting them are heterogeneous. The heterogeneity results in different capabilities for job processing and data access.

In traditional parallel and distributed systems, the computational resources are usually managed by a single control point. The scheduler not only has full information about all running/pending tasks and resource utilization, but also manages the task queue and resource pool. Thus it can easily predict the behaviours of resources, and is able to assign tasks to resources according to certain performance requirements. In a Grid, however, resources are usually autonomous and the Grid scheduler does not have full control of the resources. It cannot violate local policies of resources, which makes it hard for the Grid scheduler to estimate the exact cost of executing a task on different sites. The autonomy also results in the diversity in local resource management and access control policies, such as, for example, the priority settings for different applications and the resource reservation methods. Thus, a Grid scheduler is required to be adaptive to different local policies. The heterogeneity and autonomy on the Grid user side are represented by various parameters, including application types, resource requirements, performance models, and optimization objectives. In this situation, new concepts such as application-level scheduling and Grid economy [20] are proposed and applied for Grid scheduling.

- **Performance Dynamism***

Making a feasible scheduling usually depends on the estimate of the performance that candidate resources can provide, especially when the algorithms are static. Grid schedulers work in a dynamic environment where the performance of available resources is constantly changing. The change comes from site autonomy and the competition by applications for resources. Because of resource autonomy, usually Grid resources are not dedicated to a Grid application. For example, a Grid job submitted remotely to a computer cluster might be interrupted by a cluster's internal job which has a higher priority; new resources may join which can provide better services; or some other resources may become unavailable. The same problem happens to networks connecting Grid resources: the available bandwidth can be heavily affected by Internet traffic flows which are non-relevant to Grid jobs. For a Grid application, this kind of contention results in performance fluctuation, which makes it a hard job to evaluate the Grid scheduling performance under classic performance models. From the point view of job scheduling, performance fluctuation might be the most important characteristic of Grid computing compared with traditional systems. A feasible scheduling algorithm should be able to be adaptive to such dynamic behaviors. Some other measures are also provided to mitigate the impact of this problem, such as QoS negotiation, resource reservation (provided by the underlying resource management system) and rescheduling. We discuss algorithms related to these mechanisms in Section 3.

- **Resource Selection and Computation-Data Separation**

In traditional systems, executable codes of applications and input/output data are usually in the same site, or the input sources and output destinations are determined before the application is submitted. Thus the cost for data staging can be neglected or the cost is a constant determined before execution, and scheduling algorithms need not consider it. But in a Grid which consists of a large number of heterogeneous computing sites (from supercomputers to desktops) and storage sites connected via wide area networks, the computation sites of an application are usually selected by the Grid scheduler according to resource status and certain performance models. Additionally, in a Grid, the communication bandwidth of the underlying network is limited and shared by a host of background loads, so the inter-domain communication cost cannot be neglected. Further,

* We use the term *Dynamism* in this paper to refer to the dynamic change in grid resource performance provided to a grid application.

many Grid applications are data intensive, so the data staging cost is considerable. This situation brings about the computation-data separation problem: the advantage brought by selecting a computational resource that can provide low computational cost may be neutralized by its high access cost to the storage site.

These challenges depict unique characteristics of Grid computing, and put significant obstacles to design and implement efficient and effective Grid scheduling systems.
It is believed, however, that research achievements on traditional scheduling problems can still provide stepping-stones when a new generation of scheduling systems is being constructed [8].

3. Grid Scheduling Algorithms: State of the Art

It is well known that the complexity of a general scheduling problem is NP-Complete [42]. As mentioned in Section 1, the scheduling problem becomes more challenging because of some unique characteristics belonging to Grid computing. In this section, we provide a survey of scheduling algorithms in Grid computing, which will form a basis for future discussion of open issues in the next section.

3.1 A Taxonomy of Grid Scheduling Algorithms

In [24], Casavant et al propose a hierarchical taxonomy for scheduling algorithms in general-purpose parallel and distributed computing systems. Since Grid is a special kind of such systems, scheduling algorithms in Grid fall into a subset of this taxonomy. From the top to the bottom, this subset can be identified as what follows.

- **Local vs. Global**

At the highest level, a distinction is drawn between local and global scheduling. The local scheduling discipline determines how the processes resident on a single CPU are allocated and executed; a global scheduling policy uses information about the system to allocate processes to multiple processors to optimize a system-wide performance objective. Obviously, Grid scheduling falls into the global scheduling branch.

- **Static vs. Dynamic**

The next level in the hierarchy (under the global scheduling) is a choice between static and dynamic scheduling. This choice indicates the time at which the scheduling decisions are made. In the case of static scheduling, information regarding all resources in the Grid as well as all the tasks in an application is assumed to be available by the time the application is scheduled. By contrast, in the case of dynamic scheduling, the basic idea is to perform task allocation on the fly as the application executes. This is useful when it is impossible to determine the execution time, direction of branches and number of iterations in a loop as well as in the case where jobs arrive in a real-time mode. These variances introduce forms of non-determinism into the running program [42]. Both static and dynamic scheduling are widely adopted in Grid computing. For example, static scheduling algorithms are studied in [17] [23] and [118], and in [68] [106] [28] and [79], dynamic scheduling algorithms are presented.

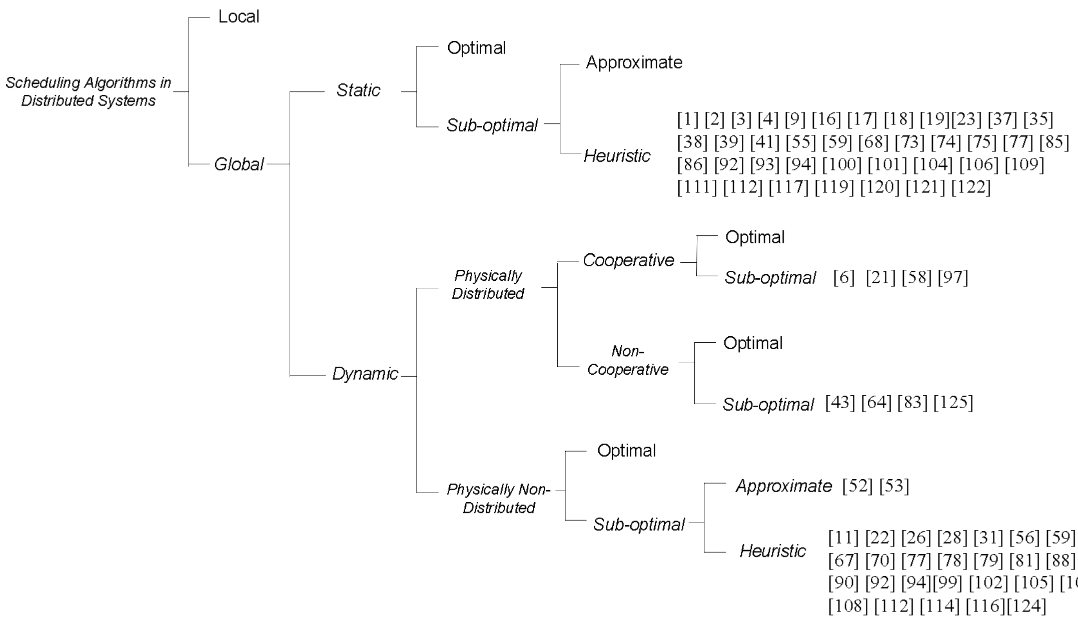

Fig. 2: A Hierarchical taxonomy for scheduling algorithms. Branches covered by Grid scheduling algorithms up to date are denoted in italics. Examples of each covered branch are shown at the leaves.

- *Static Scheduling*

In the static mode, every task comprising the job is assigned once to a resource. Thus, the placement of an application is static, and a firm estimate of the cost of the computation can be made in advance of the actual execution. One of the major benefits of the static model is that it is easier to program from a scheduler's point of view. The assignment of tasks is fixed a priori, and estimating the cost of jobs is also simplified. The static model allows a "global view" of tasks and costs. Heuristics can be used to decide whether to incur slightly higher processing costs in order to keep all the tasks involved in a job on the same or tightly-coupled nodes, or to search for lower computational costs and be penalized with slightly higher communication costs (refer to 3.4.2). But cost estimate based on static information is not adaptive to situations such as one of the nodes selected to perform a computation fails, becomes isolated from the system due to network failure, or is so heavily loaded with jobs that its response time becomes longer than expected. Unfortunately, these situations are quite possible and beyond the capability of a traditional scheduler running static scheduling policies. To alleviate this problem, some auxiliary mechanisms such as rescheduling mechanism [31] are introduced at the cost of overhead for task migration. Another side-effect of introducing these measures is that the gap between static scheduling and dynamic scheduling becomes less important [52].

- *Dynamic Scheduling*

Dynamic scheduling is usually applied when it is difficult to estimate the cost of applications, or jobs are coming online dynamically (in this case, it is also called online scheduling). A good example of these scenarios is the job queue management in some metacomputing systems like Condor [112] and Legion [26]. Dynamic task scheduling has two major components [87]: system state estimation (other than cost estimation in static scheduling) and decision making. System state estimation involves collecting state information throughout the Grid and constructing an estimate. On the basis of the estimate, decisions are made to assign a task to a selected resource. Since the cost for an assignment is not available, a natural way to keep the whole system health is balancing the loads of all resources. The advantage of dynamic load balancing over static scheduling is that the system need not be aware of the run-time behavior of the application before execution. It is particularly useful in a system where the primary performance goal is maximizing resource utilization, rather than minimizing runtime for individual jobs [64]. If a resource is assigned too many tasks, it may invoke a balancing policy to decide whether to transfer some tasks to other resources, and which tasks to transfer. According to who will initiate the balancing process, there are two different approaches: sender-initiated where a node that receives a new task but doesn't want to run the task initiates the task transfer, and receiver-initiated where a node that is willing to receive a new task initiates the process [95]. According to how the dynamic load balancing is achieved, there are four basic approaches [42]:

- Unconstrained First-In-First-Out (FIFO, also known as First-Come-First- Served)
- Balance-constrained techniques
- Cost-constrained techniques
- Hybrids of static and dynamic techniques

Unconstrained FIFO: In the unconstrained FIFO approach, the resource with the currently shortest waiting queue or the smallest waiting queue time is selected for the incoming task. This policy is also called opportunistic load balancing (OLB) [76] or myopic algorithm. The major advantage of this approach is its simplicity, but it is often far from optimal.

Balance-constrained: The balance-constrained approach attempts to rebalance the loads on all resources by periodically shifting waiting tasks from one waiting queue to another. In a massive system such as the Grid, this could be very costly due to the considerable communication delay. So some adaptive local rebalancing heuristic can be applied. For example, tasks are initially distributed to all resources, and then, instead of computing the global rebalance, the rebalancing only happens inside a "neighborhood" where all resources are better interconnected. This approach has several advantages: the initial loads can be quickly distributed to all resources and started quickly; the rebalancing process is distributed and scalable; and the communication delay of rebalancing can be reduced since task shifting only happens among the resources that are "close" to each other. A similar algorithm is proposed and evaluated in [28].

Cost-constrained: An improved approach to balance-constrained scheduling is cost-constrained scheduling. This approach not only considers the balance among resources but also the communication cost between tasks. Instead of doing a task exchange periodically, tasks will be checked before their move. If the communication cost brought by a task shift is greater than the decrease in execution time, the task will not move; otherwise, it will be moved. This approach is more efficient than the previous one when the communication costs among resources are heterogeneous and the communication cost to execute the application is the main consideration. It is also flexible, and can be used with other cost factors such as seeking lowest memory size or lowest disc drive activity, and so on. In [68], the authors propose a similar scheduling policy, but the objective of the rescheduling phase in their case is not for load balancing and cost optimizing, but rather to make running jobs release required resources for pending jobs.

Hybrid: A further improvement is the static-dynamic hybrid scheduling. The main idea behind hybrid techniques is to take the advantages of static schedule and at the same time capture uncertain behaviors of applications and resources. For the scenario of an application with uncertain behavior, static scheduling is applied to those parts that always execute. At run time, scheduling is done using statically computed estimates that reduce run-time overhead. That is, static scheduling is done on the always-executed-tasks, and dynamic scheduling on others. For example, in those cases where there are special QoS requirements in some tasks, the static phase can be used to map those tasks with QoS requirements, and dynamic scheduling can be used for the remaining tasks. For the scenario of low predictable resource behaviors, static scheduling is used to initiate task assignment at the beginning and dynamic balancing is activated when the performance estimate on which the static scheduling is based fails. Spring et al show an example of this scenario in [100].

Some other dynamic online scheduling algorithms, such as those in [2] and [77], consider the case of resource reservation which is popular in Grid computing as a way to get a degree of certainty in resource performance. Algorithms in these two examples aim to minimize the makespan of incoming jobs which consist of a set of tasks. Mateescu [77] uses a resource selector to find a co-reservation for jobs requiring multiple resources. The job queue is managed by a FIFO fashion with dynamic priority correction. If co-reservation fails for a job in a scheduling cycle, the job's priority will be promoted in the queue for the next scheduling round. The resource selector ranks candidate resources by their number of processors and memory size. Aggarwal's method [2] is introduced in subsection 3.3.

- **Optimal vs. Suboptimal**

In the case that all information regarding the state of resources and the jobs is known, an optimal assignment could be made based on some criterion function, such as minimum makespan and maximum resource utilization. But due to the NP-Complete nature of scheduling algorithms and the difficulty in Grid scenarios to make reasonable assumptions which are usually required to prove the optimality of an algorithm, current research tries to find suboptimal solutions, which can be further divided into the following two general categories.

- **Approximate vs. Heuristic**

The approximate algorithms use formal computational models, but instead of searching the entire solution space for an optimal solution, they are satisfied when a solution that is sufficiently "good" is found. In the case where a metric is available for evaluating a solution, this technique can be used to decrease the time taken to find an acceptable schedule. The factors which determine whether this approach is worthy of pursuit include [24]:

- Availability of a function to evaluate a solution.
- The time required to evaluate a solution.
- The ability to judge the value of an optimal solution according to some metric.
- Availability of a mechanism for intelligently pruning the solution space.

If traditional evaluating metrics are used for task scheduling in Grid computing, e.g., makespan, the dynamic nature of Grid computing will violate the above conditions (see 3.2), so that there are no such approximation algorithms known to date. The only approximate algorithms in Grid scheduling at the time of this writing are based on a newly proposed objective function: Total Processor Cycle Consumption [50] [51].

The other branch in the suboptimal category is called heuristic. This branch represents the class of algorithms which make the most realistic assumptions about a priori knowledge concerning process and system loading characteristics. It also represents the solutions to the scheduling problem which cannot give optimal answers but only require the most reasonable amount of cost and other system resources to perform their function. The evaluation of this kind of solution is usually based on experiments in the real world or on simulation. Not restricted by formal assumptions, heuristic algorithms are more adaptive to the Grid scenarios where both resources and applications are highly diverse and dynamic, so most of the algorithms to be discussed in the following are heuristics.

- **Distributed vs. Centralized**

In dynamic scheduling scenarios, the responsibility for making global scheduling decisions may lie with one centralized scheduler, or be shared by multiple distributed schedulers. In a computational Grid, there might be many applications submitted or required to be rescheduled simultaneously. The centralized strategy has the advantage of ease of implementation, but suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck. For example, Sabin et al [88] propose a centralized metasheduler which uses backfill to schedule parallel jobs in multiple heterogeneous sites. Similarly, Arora et al [6] present a completely decentralized, dynamic and sender-initiated scheduling and load balancing algorithm for the Grid environment. A property of this algorithm is that it uses a smart search strategy to find partner nodes to which tasks can migrate. It also overlaps this decision making process with the actual execution of ready jobs, thereby saving precious processor cycles.

- **Cooperative vs. Non-cooperative**

If a distributed scheduling algorithm is adopted, the next issue that should be considered is whether the nodes involved in job scheduling are working cooperatively or independently (non-cooperatively). In the non-cooperative case, individual schedulers act alone as autonomous entities and arrive at decisions regarding their own optimum objects independent of the effects of the decision on the rest of system. Good examples of such schedulers in the Grid are application-level schedulers which are tightly coupled with particular applications and optimize their private individual objectives.

In the cooperative case, each Grid scheduler has the responsibility to carry out its own portion of the scheduling task, but all schedulers are working toward a common system-wide goal. Each Grid scheduler's local policy is concerned with making decisions in concert with the other Grid schedulers in order to achieve some global goal, instead of making decisions which will only affect local performance or the performance of a particular job. An example of cooperative Grid scheduling is presented in [95], where the efficiency of sender-initiated and receiver-initiated algorithms adopted by distributed Grid schedulers is compared with that of centralized scheduling and local scheduling.

The hierarchy taxonomy classifies scheduling algorithms mainly from the system's point view, such as dynamic or static, distributed or centralized. There are still many other important aspects forming a scheduling algorithm that cannot be covered by this method. Casavant et al [24] call them *flat classification characteristics*. In this paper, we discuss the following properties and related examples which are rendered by current scheduling algorithms when they are confronted with new challenges in the Grid computing scenario: what's the goal for scheduling? Is the algorithm adaptive? Is there dependency among tasks in an application? How to deal with large volumes of input and out data during scheduling? How do QoS requirements influence the scheduling product? How does the scheduler fight again dynamism in the Grid? Finally, what new methodologies are applied to the Grid scheduling problem?

3.2 Objective Functions

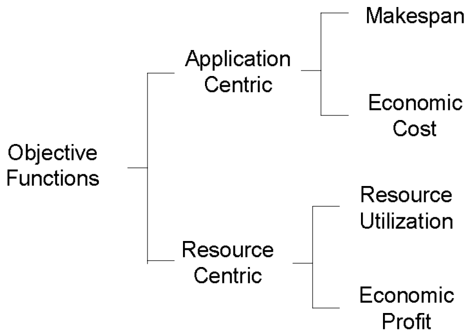

Fig. 3: Objective functions covered in this survey.

The two major parties in Grid computing, namely, resource consumers who submit various applications, and resources providers who share their resources, usually have different motivations when they join the Grid. These incentives are presented by objective functions in scheduling. Currently, most of objective functions in Grid computing are inherited from traditional parallel and distributed systems. Grid users are basically concerned with the performance of their applications, for example the total cost to run a particular application, while resource providers usually pay more attention to the performance of their resources, for example the resource utilization in a particular period. Thus objective functions can be classified into two categories: application-centric and resource-centric [123]. Fig. 3 shows the objective functions we will meet in our following discussion.

- **Application-Centric**

Scheduling algorithms adopting an application-centric scheduling objective function aim to optimize the performance of each individual application, as application-level schedulers do. Most of current Grid applications' concerns are about time, for example the *makespan*, which is the time spent from the beginning of the first task in a job to the end of the last task of the job. Makespan is the one of the most popular measurements of scheduling algorithms and many examples given in the following discussion adopt it. As economic models [18] [19] [43] [124] are introduced into Grid computing, the *economic cost* that an application needs to pay for resources utilization becomes a concern of some of Grid users. This objective function is widely adopted by Grid economic models which are mainly discussed in Subsection 3.6. Besides these simple functions, many applications use compound objective functions, for example, some want both shorter execution time and lower economic costs. The primary difficulty facing the adoption of this kind of objective functions lies in the normalization of two different measurements: time and money. Such situations make scheduling in the Grid much more complicated. It is required that Grid schedulers be adaptive enough to deal with such compound missions. At the same time, the development of the Grid infrastructure has shown a service-oriented tendency [49], so the *quality of services* (QoS) becomes a big concern of many Grid applications in such a non-dedicated dynamic environment. The meaning of QoS is highly dependent on particular applications, from hardware capacity to software existence. Usually, QoS is a constraint imposed on the scheduling process instead of the final objective function. The involvement of QoS usually has effect on the resource selection step in the scheduling process, and then influences the final objective optimization. Such scenarios will be discussed in 3.7.

- **Resource-Centric**

Scheduling algorithms adopting resource-centric scheduling objective functions aim to optimize the performance of the resources. Resource-centric objectives are usually related to resource utilization, for example, *throughput* which is the ability of a resource to process a certain number of jobs in a given period; *utilization*, which is the percentage of time a resource is busy. Low utilization means a resource is idle and wasted. For a multiprocessor resource, utilization differences among processors also describe the load balance of the system and decrease the throughput. Condor is a well known system adopting throughput as the scheduling objective [112]. As economic models are introduced into Grid computing, *economic profit* (which is the economic benefits resource providers can get by attracting Grid users to submit applications to their resources) also comes under the purview of resource management polices.

In the Grid computing environments, due to autonomy both in Grid users and resource providers, application-centric objectives and resource centric objectives often are at odds. Legion [26] provides a methodology allowing each group to express their desires, and acts as a mediator to find a resource allocation that is acceptable to both parties through a flexible, modular approach to scheduling support.

The objective functions mentioned above are widely adopted before the emergence of Grid computing and many efforts have been made to approach an approximation [27] [30] [59] [70] or to get a "good" result heuristically. But these works make a common assumption, namely, that the resources are dedicated so that they can provide constant performance to an application. But as we have emphasized in Section 2, this assumption does not hold in Grid computing. This violation weakens the previous results. For example, assume an optimal schedule with makespan *OPT* can be found, if the resources involved are stable. If the Grid resources are suddenly slowed down at *OPT* due to some reason (interrupted by resources' local jobs, network contention, or whatever) and the slow speed situation continues for a long period, then the makespan of the actual schedule is far from *OPT* and cannot be bounded by scheduling algorithms that cannot predict the performance changing. So, if the objective function of a schedule is makespan, and there is no bound either for resource performance or for the time period of the change, in other words, if we cannot predict the performance fluctuations, there will be no makespan approximation algorithm in general that applies to a Grid [51].

To escape from this predicament, a novel scheduling criterion for Grid computing is proposed in [51]: *total processor cycle consumption* (TPCC), which is the total number of instructions the Grid could compute from the starting time of executing the schedule to the completion time. TPCC represents the total computing power consumed by an application. In this new criterion, the length of a task is the number of instructions in the task; the speed of a processor is the number of instructions computed per unit time; and processors in a Grid are heterogeneous so they have various speeds. In addition, the speed of each processor varies over time due to the contention expected in an open system. Let $s_{p,t}$ be the speed of processor p during time interval $[t, t+1)$, where t is a non-negative integer. Without loss of generality, it can be assumed that the speed of each processor does not vary during time interval $[t, t+1)$, for every t by adopting an interval as short as the unit time. It is also assumed that the value of any $s_{p,t}$ cannot be known in advance. Fig. 4(a) shows an example of a set of tasks. Fig. 4(b) shows an example of processor speed distribution.

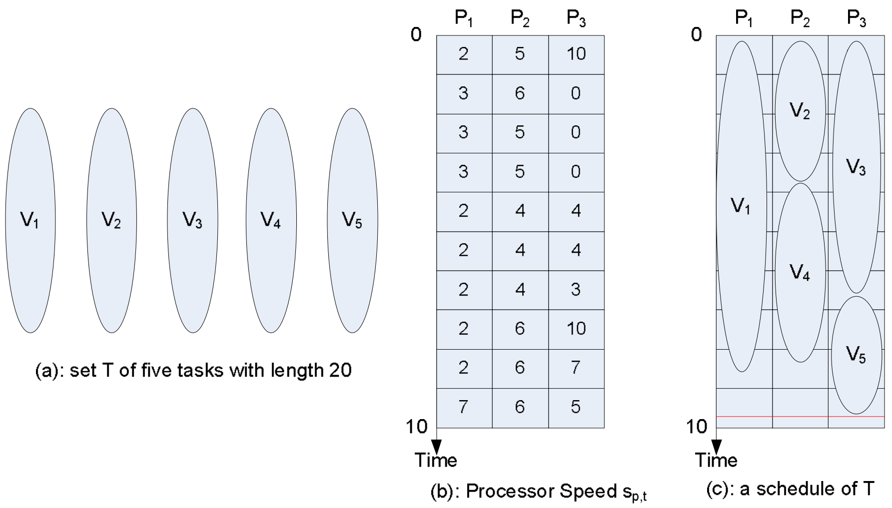

Fig. 4: A new Grid scheduling criterion: TPCC [50].

Let T be a set of n independent tasks with the same length L. Let S be a schedule of T in a Grid with m processors. Let M be the makespan of S. The speed of processor p during the time interval $[t, t+1)$ is $s_{p,t}$. Then, the TPCC of S is defined as:

$$\sum_{p=1}^{m}\sum_{t=0}^{\lfloor M \rfloor -1} s_{p,t} + \sum_{p=1}^{m}(M - \lfloor M \rfloor) s_{p,\lfloor M \rfloor} .$$

So the TPCC of the schedule in Fig. 4(c) is 21 + 45 + 38 + 7 × 2/5 + 6 × 2/5 + 5 × 2/5 = 111.2.

The advantage of TPCC is that it can be little affected by the variance of resource performance, yet still related to the makespan. Since the total number of instructions needed to run a job is constant, approximation algorithms based on this criterion can be developed. In [50], a $(1+\frac{m(\log_e(m-1)+1}{n})$ approximation algorithm is given for coarse-grained independent tasks in the Grid. A $(1+\frac{L_{co}(n)\times m(\log_e(m-1)+1}{n})$ approximation algorithm for scheduling of coarse-grained tasks with precedence orders is described in [51], where $L_{cp}(n)$ is the critical path of the task graph.

3.3 Adaptive Scheduling

An adaptive solution to the scheduling problem is one in which the algorithms and parameters used to make scheduling decisions change dynamically according to the previous, current and/or future resource status [24]. In Grid computing, the demand for scheduling adaptation comes from three points: the heterogeneity of candidate resources, the dynamism of resource performance, and the diversity of applications, as Fig. 4 shows. Correspondent with these three points, we can find three kinds of examples also.

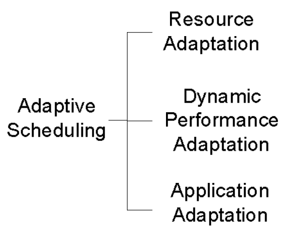

Fig. 5: Taxonomy for adaptive scheduling algorithms in Grid computing.

Resource Adaptation: Because of resource heterogeneity and application diversity, discovering available resources and selecting an application-appropriate subset of those resources are very important to achieve high performance or reduce the cost, for example Su et al [102] show how the selection of a data storage site affects the network transmission delay. Dail et al [34] propose a resource selection algorithm in which available resources are grouped first into disjoint subsets according to the network delays between the subsets. Then, inside each subset, resources are ranked according to their memory size and computational power. Finally, an appropriately-sized resource group is selected from the sorted lists. The upper bound for this exhaustive search procedure is given and claimed acceptable in the computational Grid circumstance. Subhlok et al [104] show algorithms to jointly analyze computation and communication resources for different application demands and a framework for automatic node selection. The algorithms are adaptive to demands like selecting a set of nodes to maximize the minimum available bandwidth between any pair of nodes and selecting a set of nodes to maximize the minimum available fractional compute and communication capacities. The complexity of these algorithms is also analyzed and the results have shown it is insignificant in comparison with the execution time of the applications that they are applied to.

Dynamic Performance Adaptation: The adaptation to the dynamic performance of resources is mainly exhibited as: (i) changing scheduling policies or rescheduling [100] [81] (for example, the switching between static scheduling algorithms which use predicted resource information and dynamic ones which balance the static scheduling results), (ii) workload distributing according to application-specific performance models [11], and (iii) finding a proper number of resources to be used [22] [57]. Applications to which these adaptive strategies are applied usually adopt some kind of divide-and-conquer approach to solve a certain problem [81]. In the divide-and-conquer approach, the initial problem can be recursively divided into sub-problems which can be solved more easily. As a special case of the divide-and-conquer approach, a model for applications following the manager/worker model is shown in Fig. 6 [63]. From an initial task (node A in Fig. 6), a number of tasks (nodes B, C, D and E) are launched to execute on pre-selected or dynamically assigned resources. Each task may receive a discrete set of data, and fulfil its computational task independently and deliver its output (Node F). Examples of such applications include parameter sweep applications [22] [23] [60], and data stripe processing [11] [100]. In contrast with the manager/worker model (where the manager is in charge of the behaviors of its workers), an active adaptation method named Cluster-aware Random Stealing (CRS) for Grid computing systems is proposed in [81] based on the traditional Random Stealing (RS) algorithm. CRS allows an idle resource steal jobs not only from the local cluster but also from remote ones with a very limited amount of wide-area communication. Thus, load balancing among nodes running a divide-and-conquer application is achieved. In reviewing experiences gained for application level scheduling in Grid computing, Berman et al [13] note that via schedule adaptation, it is possible to use sophisticated scheduling heuristics, like list-scheduling approaches which are sensitive to performance prediction errors, for Grid environments in which resource availabilities change over time.

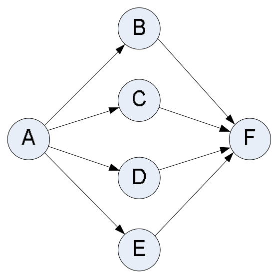

Fig. 6: The parallel workflow of a divide-and-conquer application.

Application Adaptation: To achieve high performance, application-level schedulers in the Grid (e.g. AppLeS [13]) are usually tightly integrated with the application itself and are not easily applied to other applications. As a result, each scheduler is application-specific. Noticing this limitation, Dail et al [34] explicitly decouple the scheduler core (the searching procedure introduced in the beginning of this subsection) from application-specific (e.g. performance models) and platform-specific (e.g. resource information collection) components used by the core. The key feature to implement the decoupling (while still keeping awareness of application characteristics) is that application characteristics are recorded and/or discovered by components such as a specialized compiler and Grid-enabled libraries. These application characteristics are communicated via well-defined interfaces to schedulers so that schedulers can be general-purpose, while still providing services that are appropriate to the application at hand. Aggarwal et al [2] consider another case that applications in Grid computing often meet, namely, resource reservation, and develop a generalized Grid scheduling algorithm that can efficiently assign jobs having arbitrary inter-dependency constraints and arbitrary processing durations to resources having prior reservations. Their algorithm also takes into account arbitrary delays in transfer of data from the parent tasks to the child tasks. In fact, this is a heuristic list algorithm which we will discus in the next subsection. In [113], Wu et al give a very good example of how a self-adaptive scheduling algorithm cooperates with long-term resource performance prediction [54] [105]. Their algorithm is adaptive to indivisible single sequential jobs, jobs that can be partitioned into independent parallel tasks, and jobs that have a set of indivisible tasks. When prediction error of the system utilization is reaching a threshold, the scheduler will try to reallocate tasks.

3.4 Task Dependency of an Application

When the relations among tasks in a Grid application are considered, a common dichotomy used is dependency vs. independency. Usually, dependency means there are precedence orders existing in tasks, that is, a task cannot start until all its parent are done. Dependency has crucial impact to the design of scheduling algorithms, so in this subsection, algorithms are discussed by following the same dichotomy as shown in Fig. 7.

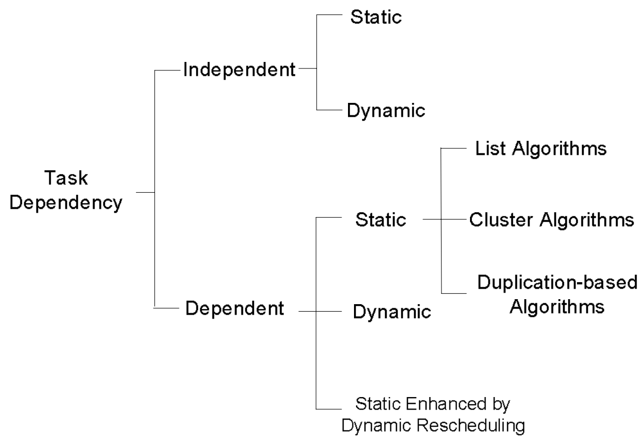

Fig. 7: Task dependency taxonomy of Grid scheduling algorithms.

3.4.1 Independent Task Scheduling

As a set of independent tasks arrive, from the system's point view, a common strategy is to assign them according to the load of resources in order to achieve high system throughput. This approach was discussed under the dynamic branch in Subsection 3.1. From the point of view of applications, some static heuristic algorithms based on execution cost estimate can be applied [17].

- **Examples of Static Algorithms with Performance Estimate**

MET (Minimum Execution Time): MET assigns each task to the resource with the best expected execution time for that task, no matter whether this resource is available or not at the present time. The motivation behind MET is to give each task its best machine. This can cause a severe load imbalance among machines. Even worse, this heuristic is not applicable to heterogeneous computing environments where resources and tasks are characterized as consistent, which means a machine that can run a task faster will run all the other tasks faster.

MCT (Minimum Completion Time): MCT assigns each task, in an arbitrary order, to the resource with the minimum expected completion time for that task. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The intuition behind MCT is to combine the benefits of opportunistic load balancing (OLB) and MET, while avoiding the circumstances in which OLB and MET perform poorly.

Min-min: The Min-min heuristic begins with the set U of all unmapped tasks. Then, the set of minimum completion time M for each task in U is found. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine (hence the name Min-min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty). Min-min is based on the minimum completion time, as is MCT. However, Min-min considers all unmapped tasks during each mapping decision and MCT only considers one task at a time. Min-min maps the tasks in the order that changes the machine availability status by the least amount that any assignment could. Let t_i be the first task mapped by Min-min onto an empty system. The machine that finishes t_i the earliest, say m_j, is also the machine that executes t_i the fastest. For every task that Min-min maps after t_i, the Min-min heuristic changes the availability status of m_j by the least possible amount for every assignment. Therefore, the percentage of tasks assigned to their first choice (on the basis of execution time) is likely to be higher for Min-min than for Max-min (see below). The expectation is that a smaller makespan can be obtained if more tasks are assigned to the machines that complete them the earliest and also execute them the fastest.

Max-min: The Max-min heuristic is very similar to Min-min. It also begins with the set U of all unmapped tasks. Then, the set of minimum completion time M, is found. Next, the task with the overall maximum from M is selected and assigned to the corresponding machine (hence the name Max-min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty). Intuitively, Max-min attempts to minimize the penalties incurred from performing tasks with longer execution times. Assume, for example, that the job being mapped has many tasks with very short execution times and one task with a very long execution time. Mapping the task with the longer execution time to its best machine first allows this task to be executed concurrently with the remaining tasks (with shorter execution times). For this case, this would be a better mapping than a Min-min mapping, where all of the shorter tasks would execute first, and then the longer running task would be executed while several machines sit idle. Thus, in cases similar to this example, the Max-min heuristic may give a mapping with a more balanced load across machines and a better makespan.

Min-min and Max-min algorithms are simple and can be easily amended to adapt to different scenarios. For example, in [57], a QoS Guided Min-min heuristic is presented which can guarantee the QoS requirements of particular tasks and minimize the makespan at the same time. Wu, Shu and Zhang [115] gave a Segmented Min-min algorithm, in which tasks are first ordered by the expected completion time (it could be the maximum ECT, minimum ECT or average ECT on all of the resources), then the ordered sequence is segmented, and finally Min-min is applied to all these segments. The segment improves the performance of typical Min-min when the lengths of the tasks are dramatically different by giving a chance to longer tasks to be executed earlier than in the case where the typical Min-min is adopted.

XSuffrage: Another popular heuristic for independent scheduling is called *Suffrage* [76]. The rationale behind Suffrage is that a task should be assigned to a certain host and if it does not go to that host, it will suffer the most. For each task, its suffrage value is defined as the difference between its best MCT and its second-best MCT. Tasks with high suffrage value take precedence. But when there is input and output data for the tasks, and resources are clustered, conventional suffrage algorithms may have problems. In this case, intuitively, tasks should be assigned to the resources as near as possible to the data source to reduce the makespan. But if the resources are clustered, and nodes in the same cluster are with near identical performance, then the best and second best MCTs are also nearly identical which makes the suffrage close to zero and gives the tasks low priority. Other tasks might be assigned on these nodes so that the task might be pushed away from its data source. To fix this problem, Casanova et al gave an improvement called *XSuffrage* in [23] which gives a cluster level suffrage value to each task. Experiments show that XSuffrage outperforms the conventional Suffrage not only in the case where large data files are needed, but also when the resource information cannot be predicted very accurately.

Task Grouping: The above algorithms are usually used to schedule applications that consist of a set of independent coarse-grained compute-intensive tasks. This is the ideal case for which the computational Grid was designed. But there are some other cases in which applications with a large number of lightweight jobs. The overall processing of these applications involves a high overhead cost in terms of scheduling and transmission to or from Grid resources. Muthuvelu et al [79] propose a dynamic task grouping scheduling algorithm to deal with these cases. Once a set of fine grained tasks are received, the scheduler groups them according to their requirements for computation (measured in number of instructions) and the processing capability that a Grid resource can provide in a certain time period. All tasks in the same group are submitted to the same resource which can finish them all in the given time. By this mean, the overhead for scheduling and job launching is reduced and resource utilization is increased.

The Problem of Heterogeneity: In heterogeneous environments, the performance of the above algorithms is also affected by the rate of heterogeneity of the tasks and the resources as well as the consistency of the tasks' estimated completion time on different machines. The study in [76] shows that no single algorithm can have a permanent advantage in all scenarios. This result clearly leads to the conclusion that if high performance is wanted as much as possible in a computational Grid, the scheduler should have the ability to adapt different application/resource heterogeneities.

- **Algorithms without Performance Estimate**

The algorithms introduced above use predicted performance to make task assignments. In [103] and [97] two algorithms are proposed that do not use performance estimate but adopt the idea of duplication, which is feasible in the Grid environment where computational resources are usually abundant but mutable.

Subramani et al [103] introduce a simple distributed duplication scheme for independent job scheduling in the Grid. A Grid scheduler distributes each job to the K least loaded sites. Each of these K sites schedules the job locally. When a job is able to start at any of the sites, the site informs the scheduler at the job-originating site, which in turn contacts the other K-1 sites to cancel the jobs from their respective queues. By placing each job in the queue at multiple sites, the expectations are improved system utilization and reduced average job makespan. The parameter K can be varied depending upon the scalability required. Silva et al [97] propose a resource information free algorithm called *Workqueue with Replication* (WQR) for independent job scheduling in the Grid. The WQR algorithm uses task replication to cope with the heterogeneity of hosts and tasks, and also the dynamic variation of resource availability due to load generated by others users in the Grid. Unlike the scheme in [103] where there are no duplicated tasks actually running, in WQR, an idle resource will replicate tasks that are still running on other resources. Tasks are replicated until a predefined maximum number of replicas are reached. When a task replica finishes, other replicas are cancelled. In this approach, performance is increased in situations when tasks are assigned to slow/busy hosts because when a task is replicated, there is a greater chance that a replica is assigned to a fast/idle host. Another advantage of this scheme is that it increases the immunity to performance changing, since the possibility that all sites are changing is much smaller than one site.

3.4.2 Dependent Task Scheduling

When tasks composing a job have precedence orders, a popular model applied is the directed acyclic graph (DAG), in which a node represents a task and a directed edge denotes the precedence orders between its two vertices. In some cases, weights can be added to nodes and edges to express computational costs and communicating costs respectively. As Grid computing infrastructures become more and more mature and powerful, support for complicated workflow applications, which can be usually modeled by DAGs, are provided. We can find such tools like Condor DAGMan [127], CoG [74], Pegasus [37], GridFlow [21] and ASKALON [110]. A comprehensive survey of these systems is given in [121], while we continue our focus on their scheduling algorithm components in what follows.

- **Grid Systems Supporting Dependent Task Scheduling**

To run a workflow in a Grid, we need to consider two problems: 1) how the tasks in the workflow are scheduled, and 2) how to submit the scheduled tasks to Grid resources without violating the structure of the original workflow. *Grid workflow generators* address the first problem and *Grid workflow engines* are used to deal with the second.

- *Grid Workflow Engines*

Grid workflow engines are responsible for CoG is a set of APIs which can be used to submit concrete workflows to the Grid; here the concrete workflow means the tasks in a DAG are already mapped to resource locations where they are to be executed, so CoG itself does not consider the optimization problem of workflows. DAGMan works similar with CoG. It accepts DAG description files representing workflows, and then following the order of tasks and dependency constraints in the description files, submits tasks to

Condor-G, which schedule them onto the best machines available in a FIFO strategy without any long-term optimization, just like it does with common Condor tasks.

- *Grid Workflow Generators*

Pegasus provides a bridge between Grid users and workflow execution engines like DAGMan. In Pegasus, there are two kinds of workflows: abstract workflows which are composed of tasks (referred as *application components* in Pegasus) and their dependencies reflecting the data dependencies of tasks, and concrete workflows which are the mappings of abstract workflows to Grid resources. Pegasus' main concern is to generate these two kinds of workflows according to demands by users for certain data products. It does so by searching available application components which can produce the required data products and available input and intermediate data replicas in the Grid. To this ends, it provides a Concrete Workflow Generator (CWG) [36]. CWG performs the mapping from an abstract workflow to a concrete workflow and generates the correspondent DAGMan submit files. It automatically identifies physical locations for both application components and data, finds appropriate resources to execute the components relying on GIS, and generates an executable workflow that can be submitted to the Condor-G through DAGMan. When there are multiple appropriate resources available, CWG supports a few standard selection algorithms: random, round-robin and min-min [38]. Resource selection algorithms are pluggable components in Pegasus so third-party developed algorithms can be applied according to different concerns. As an example, Blythe et al [16] present a multiple rounds mixed min-min and max-min algorithm for resource selection in which the final mapping selected is the one that has the minimal makespan. Considering the dynamism of the Grid, instead of submitting the whole task graph at once, Pegasus applies a workflow partition method that submits layer-partitioned subgraphs iteratively. But, as shown in the discussion below, layered partition may not use the advantages of locality for task independency and, as a result, produce bad schedules, especially when the DAG is unbalanced. This weakness is also demonstrated in [110].

Similar with Pegasus, **ICENI** [78] also adopts pluggable algorithms for abstract workflow to concrete workflow mapping, and in [119], random, best of *n* random, simulated annealing and game theory algorithms are tested. The latter two algorithms will be discussed in Subsection 3.6.

In **GridFlow** [21], workflow scheduling is conducted hierarchically by a global Grid workflow manager and a local Grid sub-workflow scheduler. Global Grid workflow manager receives requests from users with the workflow description in XML, and then simulates workflow execution to find a near-optimal schedule in terms of makespan. The simulation is done by polling local Grid schedulers which can estimate the finish time of sub-workflows on their local sites. A fuzzy timing technique is used to get the estimate, and the possibility of conflict on a shared resource among tasks from different sub-workflows is considered. The advantage of fuzzy functions is that they can be computed very fast and are suitable for scheduling of time-critical Grid applications, although they do not necessarily provide the best scheduling solution. GridFlow also provides rescheduling functionality when the real execution is delayed too far from the estimate.

We see from the above discussion that most current efforts have been directed towards supporting workflows at the programming level, thus providing potential opportunities for algorithms designers (as they allow scheduling algorithms to be plugged in). As Grid computing inherits problems from traditional systems, a natural question to ask is what can be learned from the extensive studies on DAG scheduling algorithms in heterogeneous computing? A complete survey of these algorithms is beyond the scope of this paper, but some ideas and common examples are discussed below to show the problems we are still confronted with in the Grid.

- **Taxonomy of Algorithms for Dependent Task Scheduling**

Considering communication delays when making scheduling decisions introduces a big challenge: the trade-off between taking advantage of maximal parallelism and minimizing communication delay. This problem is also called the max-min problem [42]. High parallelism means dispatching more tasks simultaneously to different resources, thus increasing the communication cost, especially when the communication delay is very high. However, clustering tasks only on a few resources means low resource utilization. To deal with this problem in heterogeneous computing systems, three kinds of heuristic algorithms were previously proposed.

- *List Heuristics*

In general, list scheduling is a class of scheduling heuristics in which tasks are assigned with priorities and placed in a list ordered in decreasing magnitude of priority. Whenever tasks contend for processing, the selection of tasks to be immediately processed is done on the basis of priority with higher-priority tasks being assigned resources first [42]. The differences among various list heuristics mainly lie in how the priority is defined and when a task is considered ready for assignment.

An important issue in DAG scheduling is how to rank (or weigh) the nodes and edges (when communication delay is considered). The rank of a node is used as its priority in the scheduling. Once the nodes and edges are ranked, task-to-resource assignment can be found by considering the following two problems to minimize the makespan: how to parallelize those tasks having no precedence orders in the graph and how to make the time cost along with the critical path in the DAG as small as possible. Many list heurists have been invented, and some new proposals can be found in [107], [91] and [83] as well as the comparison of their algorithms with older ones.

- Two Classic Examples

HEFT: Topcuoglu et al [107] present a heuristic called Heterogeneous Earliest-Finish-Time (*HEFT*) algorithm. The HEFT algorithm selects the task with the highest upward rank (an upward rank is defined as the maximum distance from the current node to the exiting node, including the computational cost and communication cost) at each step. The selected task is then assigned to the processor which minimizes its earliest finish time with an insertion-based approach which considers the possible insertion of a task in an earliest idle time slot between two already-scheduled tasks on the same resource. The time complexity of HEFT is $O(e \times p)$, where e is the number of edges and p is the number of resources.

HEFT might be one of the most frequently referred to listing algorithms which aim to reduce makespan of tasks in a DAG. For example, it is tested in ASKALON and compared with a genetic algorithm and a myopic algorithm [110], and the results show its effectiveness in the Grid scenarios, especially when the task graph is unbalanced.

FCP: Radulescu et al [83] present another list heuristic called Fast Critical Path (FCP), intending to reduce the complexity of the list heuristics while maintaining the scheduling performance at the same time. The motivation of FCP is based on the following observation regarding the complexity of list heuristics. Basically, a list heuristic has the following procedures: the $O(e + v)$ time ranking phase, the $O(v \log v)$ time ordering phase, and finally the $O((e + v) \times p)$ time resource selecting phase, where e is the number of edges, v is the number of tasks and p is the number of resources. Usually the third term is larger than the second term. The FCP algorithm does not sort all the tasks at the beginning but maintains only a limited number of tasks sorted at any given time. Instead of considering all processors as possible targets for a given task, the choice is restricted to either the processor from which the last messages to the given task arrives or the processor which becomes idle the earliest. As a result, the time complexity is reduced to $O(v \log p + e)$.

- The Problem of Heterogeneity:

A critical issue in list heuristics for DAGs is how to compute a node's rank. In a heterogeneous environment, the execution time of the same task will differ on different resources as well as the communication cost via different network links. So for a particular node, its rank will also be different if it is assigned to different resources. The problem is how to choose the proper value used to make the ordering decision. These values could be the mean value (like the original HEFT in [107]), the median value [62], the worst value, the best value and so on. But Zhao et al [122] have shown that different choices can affect the performance of list heuristics such as HEFT dramatically (makespan can change 47.2% for certain graph). Motivated by this observation, Sakellariou et al [91] gave a *Hybrid* algorithm which is less sensitive to different approaches for ranking nodes. In this algorithm, tasks are upward ranked and sorted decreasingly. Then the sorted tasks are grouped along the sorted sequence and in every group, tasks are independent. Finally, each group can be assigned to resources using heuristics for independent tasks.

The above algorithms have explored how the heterogeneity of resources and tasks impacts the scheduling algorithm, but they only consider the heterogeneity of computational resources, and miss the heterogeneity of communication links.

- Instances of List Heuristics in Grid Computing

Previous research in DAG scheduling algorithms is very helpful when we consider the same problem in the Grid scenario. For example, a list scheduling algorithm is proposed in [118], which is similar to the HEFT algorithm, but changes the method to compute the level of a task node by not only including its longest path to an exit node, but also taking incoming communication cost from its parents into account. In [75], Ma et al propose a new list algorithm called Extended Dynamic Critical Path (xDCP) which is a Grid-enabled version of the Dynamic Critical Path (DCP) algorithm, applied in a homogenous environment. The idea behind DCP is continuous shortening the critical path in the task graph, by scheduling tasks in the current CP to a resource where a task on the critical path has an earlier start time. The xDCP algorithm was proposed for scheduling parameter-swap applications in a heterogeneous Grid. The improvements include: (i) initial shuffling tasks to multiple resources when the scheduling begins instead of keeping them on one node, (ii) using the finish time instead of start time to rank task nodes to adapt to heterogeneous resources, and (iii) multiple rounds of scheduling to improve the current scheduling instead of only scheduling once. The complexity of xDCP is $O(v^3)$, which is the same as the DCP.

o *Duplication Based Algorithms*

An alternative way to shorten the makespan is to duplicate tasks on different resources. The main idea behind duplication based scheduling is utilizing resource idle time to duplicate predecessor tasks. This may avoid the transfer of results from a predecessor to a successor, thus reducing the communication cost. So duplication can solve the max-min problem.

Duplication based algorithms differ according to the task selection strategies for duplication. Originally, algorithms in this group were usually for an unbounded number of identical processors such as distributed memory multiprocessor systems. Also they have higher complexity than the algorithms discussed above. For example, Darbha et al [35] present such an algorithm named TDS (Task Duplication-based Scheduling Algorithm) in a distributed-memory machine with a complexity of $O(v^2)$. (Note, this complexity is for homogeneous environments.)

TDS: In [35], for each node in a DAG, its *earliest start time* (*est*), *earliest completion time* (*ect*), *latest allowable start time* (*last*), *latest allowable completed time* (*lact*), and *favorite predecessor* (*fpred*) should be computed. The *last* is the latest time when a task should be started; otherwise, successors of this task will be delayed (that is, their *est* will be violated). The favorite predecessors of a node *i* are those which are predecessors of *i* and if *i* is assigned to the same processors on which these nodes are running, *est*(*i*) will be minimized. The *level* value of a node (which denotes the length of the longest path from that node to an exit node (also known as sink node), ignoring the communicating cost along that path) is used as the priority to determine the processing order of each task. To compute these values, the whole DAG of the job will be traversed, and the complexity needed for this step is $O(e + v)$. Based on these values, task clusters are created iteratively. The clustering step is like a depth-first search from an unassigned node having the lowest level value to an entry node. Once an entry node is reached, a cluster is generated and tasks in the same cluster will be assigned to the same resource. In this step, the *last* and *lact* values are used to determine whether duplication is needed. For example, if *j* is a favorite predecessor of *i* and $(last(i) - lact(j)) < c_{j,i}$, where $c_{j,i}$ is the communication cost between *j* and *i*, *i* will be assigned to the same processor as *j*, and if *j* has be assigned to other processors, it will be duplicated to *i*'s processor. In the clustering step, the DAG is traversed similarly to the depth-first search from the exiting node, and the complexity of this step would be the same as the complexity of a general search algorithm, which is also $O(v + e)$. So the overall complexity is $O(v + e)$. In a dense DAG, the number of edges is proportional to $O(v^2)$, which is the worst case complexity of duplication algorithm. Note, in the clustering step, the number of resources available is always assumed to be smaller than required, that is, the number of resources is unbounded.

TANH: To exploit the duplication idea in heterogeneous environments, a new algorithm called TANH (Task duplication-based scheduling Algorithm for Network of Heterogeneous systems) is presented in [84] and [9]. Compared to the version for homogeneous resources, the heterogeneous version has higher complexity, which is $O(v^2 \times p)$. This is reasonable since the execution time of a task differs on different resources. A new parameter is introduced for each task: the favorite processor (*fp*), which can complete the task earliest. Other parameters of a task are computed based on the value of *fp*. In the clustering step, the initial task of a cluster is assigned to its first *fp*, and if the first *fp* has already been assigned, then to the second and so on. A processor reduction algorithm is also provided in [9], which is used to merge clusters when the number for processors is less than that the clusters generate.

Duplication based algorithms are very useful in Grid environments. The computational Grid usually has abundant computational resources (recall that the number of resource is unbounded in some duplication algorithms), but high communication cost. This will make task duplication very cost effective. Duplication has already received some attention (see, for example, [103] and [97]), but current duplication based scheduling algorithms in the Grid only deal with independent jobs. There are opportunities to create new algorithms for complicated DAGs scheduling in an environment that is not only heterogeneous, but also dynamic.

o ***Clustering Heuristics***

In parallel and distributed systems, clustering is an efficient way to reduce communication delay in DAGs by grouping heavily communicating tasks to the same labeled clusters and then assigning tasks in a cluster to the same resource. In general, clustering algorithms have two phases: the task clustering phase that partitions the original task graph into clusters, and a post-clustering phase which can refine the clusters produced in the previous phase and get the final task-to-resource map.

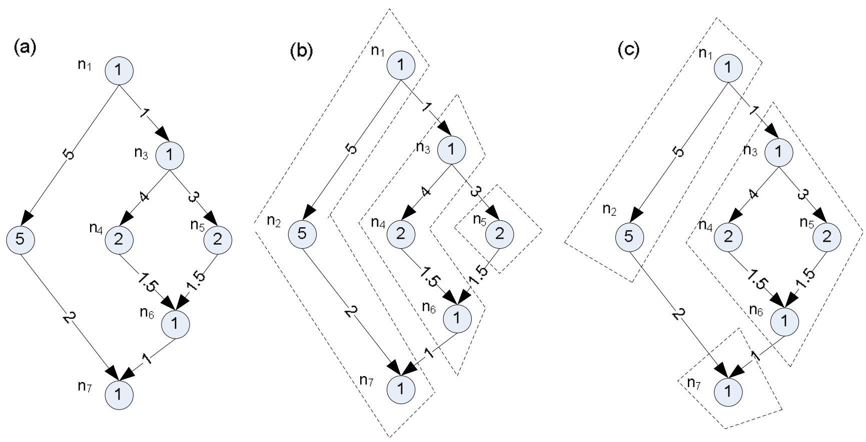

Fig. 8: (a) A DAG with computational and communication cost. (b) A linear clustering. (c) A nonlinear clustering[53].

▪ Algorithms for Task Clustering

At the beginning, each node in a task graph is an independent cluster. In each iteration, previous clusters are refined by merging some clusters. Generally, clustering algorithms map tasks in a given DAG to an unlimited number of resources. In practice, an additional cluster merging step is needed after clusters are generated, so that the number of clusters generated can be equal to the number of processors. A task cluster could be *linear* or *nonlinear*. A clustering is called nonlinear if two independent tasks are mapped in the same cluster; otherwise it is called linear. Fig. 8 shows a DAG with computational and communication cost (Fig. 8(a)), a linear clustering with three clusters $\{n_1, n_2, n_7)$, $\{n_3, n_4, n_6\}$, $\{n_5\}$ (Fig. 8(b)), and a nonlinear clustering with clusters $\{n_1, n_7\}$, $\{n_3, n_4, n_5, n_6\}$, and $\{n_7\}$ (Fig. 8(c)) [53]. The problem of obtaining an optimal clustering of a general task graph is NP-complete, so heuristics are designed to deal with this problem ([53] [117] [71] [72]).

DSC: Yang et al [117] propose a clustering heuristic called *Dominant Sequence Clustering* (DSC) algorithm. The *critical path* of a scheduled DAG is called *Dominant Sequence* (DS) to distinguish it from the *critical path* of a clustered DAG. The *critical path* of a clustered graph is the longest path in that graph, including both non-zero communication edge cost and task weights in that path. The makespan in executing a clustered DAG is determined by the Dominant Sequence, not by the critical path of the clustered DAG. Fig. 9 (a) shows the critical path of a clustered graph, which consists of $\{n_1, n_2, n_7\}$ with a length of 9. Fig. 9(b) is a schedule of this clustered graph, and Fig. 9 (c) gives the DS of the scheduled task graph, which consists of $\{n_1, n_3, n_4, n_5, n_6, n_7\}$ with a length of 10 [53].

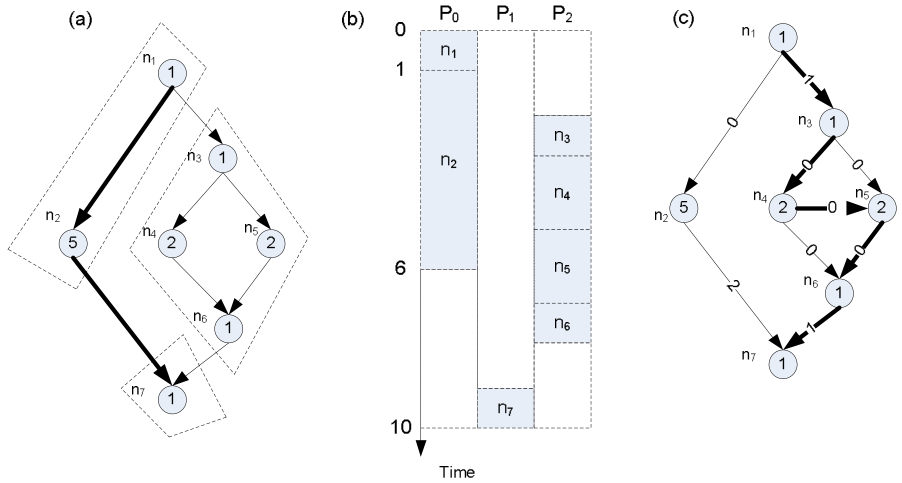

Fig. 9: (a) The clustered DAG and its CP shown in thick arrows. (b) The Gantt chart of a schedule. (c) The scheduled DAG and the DS shown in thick arrows[53].

In the DSC algorithm, task priorities are dynamically computed as the sum of their *top level* and *bottom level*. The *top level* and *bottom level* are the sum of the computation and communication costs along the longest path from the given task to an entry task and an exit task, respectively. While the bottom level is statically computed at the beginning, the top level is computed incrementally during the scheduling process. Tasks are scheduled in the order of their priorities. The current node is an unassigned node with highest propriety. Since the entry node always has the longest path to the exit node, clustering always begins with the entry node. The current node is merged with the cluster of one of its predecessors so that the *top level* value of this node can be minimized. If all possible merging increases the *top level* value, the current node will remain in its own cluster. After the current node is clustered, priorities of all its successors will be updated. The time complexity of DSC is $O((e + v) \log v)$, in which $O(\log v)$ comes from priority updating at each step using a binary heap, and $(e + v)$ is for graph traversal in the clustering iterations. So for a dense task graph, the complexity is roughly $O(v^2 \log v)$.

CASS-II: Liou et al [71] present a cluster algorithm called *CASS-II* which employs a two-step approach. In the first step, CASS-II computes for each node v a value $s(v)$, which is the length of the longest path from an entry node to v (excluding the execution time of v). Thus, $s(v)$ is the start time of v before clustering, and $s(v)$ is 0 if v is an entry node. The second step is the clustering step. Just like DSC, it consists of a sequence of refinement steps, where each refinement step creates a new cluster or ''grows'' an existing cluster. Unlike DSC, CASS-II constructs the clusters bottom-up, i.e., starting from the exit nodes. To construct the clusters, the algorithm computes for each node v a value $f(v)$, which is the longest path from v to an exit node in the current partially clustered DAG. Let $l(v) = f(v) + s(v)$. The algorithm uses $l(v)$ to determine whether the node v can be considered for clustering at the current refinement step. The clustering begins by placing every exit node in its own cluster, and then goes through a sequence of iterations. In each iteration, it considers to cluster every node v whose immediate successors have all been clustered. Node v is merged to the cluster of its successor which gives it the minimum $l(v)$ value if the merge will not increase the $l(v)$ value. CASS-II does not re-compute the critical path in each refinement step. Therefore, the algorithm has a complexity $O(e + v \log v)$, and shows 3 to 5 times faster than DSC in experiments according to [71].

- Algorithms for the Post-clustering Phase

In [72], steps after task clustering are studied, which include the cluster merging, processor assignment and task ordering in local processors. For the cluster merging, three strategies are compared, namely: load balancing (LB), communication traffic minimization (CTM), and random (RAND).

- LB: Define the (computational) *workload* of a cluster as the sum of execution times of the tasks in the cluster. At each merging step, choose a cluster, $C1$, that has a minimum workload among all clusters, and find a cluster, $C2$, that has a minimum workload among those clusters which have a communication edge between themselves and $C1$. Then the pair of clusters $C1$ and $C2$ are merged.
- CTM: Define the (communication) traffic of a pair of clusters ($C1$, $C2$) as the sum of communication times of the edges from $C1$ to $C2$ and from $C2$ to $C1$. At each merging step, merge the pair of clusters which have the most traffic.
- RAND: At each refinement step, merge a random pair of clusters.

For the processor assignment, a simple heuristic is applied to find a one-to-one mapping between a cluster and a processor: (1) assign the cluster with the largest total communication traffic with all other clusters to a processor; (2) choose an unassigned cluster having the largest communication traffic with an assigned cluster and place it in a processor closest to its communicating partner; (3) repeat (2) until all clusters have been assigned to processors.

Experimental results in [71] and [72] indicate that the performance of clustering heuristics evaluated by makespan depends on the *granularity* of the tasks of a graph. The *granularity* of a task is the ratio of its execution time vs. the overhead incurred when communicating with other tasks. This result means adaptive ability will be helpful for the scheduler to provide higher scheduling quality if jobs have high diversity.

- The Problem of Heterogeneity

According to the basic idea of task clustering, cluster heuristics need not consider heterogeneity of the resources in the clustering phase. But in the following cluster merging and resource assigning phases, heterogeneity will definitely affect the final performance. Obviously, research in [72] does not consider this problem and no other research on this problem has been performed, to our knowledge. Cluster heuristics have not yet improved for Grid computing either, where communication is usually costly and performance of resources varies over time. Therefore this remains an interesting topic for research in the Grid computing environement. Another value of the cluster heuristic for Grid scheduling is its multi-phase nature, which provides more flexibility to the Grid scheduler to employ different strategies according to the configuration and organization of the underlying resources.

- **Algorithms Considering Dynamism of Grid**

However, there is an important issue for Grid computing which has not been discussed: the resource performance dynamism. All algorithms that we have mentioned in this subsection schedule whole task graphs on the basis of static resource performance estimate, which could be jeopardized by resource performance change during the execution period. Usually the performance dynamism is resulted from completion among jobs sharing the same resource. This problem could be reconciled by considering the possibility of conflict when the scheduling decision is made. He et al [56] show us an example of this approach. Their algorithm considers the optimization of DAG makespan on multiclusters which have their own local schedulers and queues shared by other background workloads, which arrive as a linear function of time. The motivation is to map as many tasks as possible to the same cluster in order to fully utilize the parallel processing capability, and at the same time reduce the inter-cluster communication. The schedulers have a hierarchical structure: the global scheduler is responsible for mapping tasks to different clusters according to their latest finish time in order to minimize the excess over the length of critical path. The local scheduler on each multicluster provides the estimated finish time of a particular task on this cluster, reports it to the global scheduler upon queries, and manages its local queue in a FIFO way. The time complexity of the global mapping algorithm is $O(p*(n+1)*n/2+e)$, where p is the number of multiclusters.

Another approach to deal with the dynamic problem is applying dynamic algorithms. In [75], the authors propose a pM-S algorithms which extends a traditional dynamic Master-Slave scheduling model. In the pM-S algorithm, two queues are used by the master, the *unscheduled* queue and the *ready* queue. Only tasks in the Ready queue can be directly dispatched to slave nodes, and tasks in the unscheduled queue can be only put into the ready queue when all its parents have been in the ready queue or dispatched. The dispatching order in the ready queue is based on tasks' priorities. When a task is finished, the priorities of all its children's ancestors will be dynamically promoted.

In [62], another dynamic algorithm is proposed for scheduling DAGs in a shared heterogeneous distributed system. Unlike the previous works in which a unique global scheduler exists, in this work, the authors consider multiple independent schedulers which have no knowledge about other jobs. So there is the danger of conflicts on resources. This algorithm is derived from a static list algorithm: the Dynamic Level Scheduling (DLS). In the original DLS, the dynamic level of a task in a DAG is used to adapt to the heterogeneity in resources, while in the newly proposed algorithm, the dynamic length of the queue on each resource is also taken into account for computing a task's level. To estimate the length of the queue, it is assumed that jobs are coming following a Poisson distribution. The research also discusses how to choose the time when the scheduling decision is made and the time when a task should be put into resources local queue in a dynamic system. Ready task queuing probabilities which are computed following the Poisson distribution are used to make these decisions.

All DAG scheduling algorithms we have discussed so far have the same goal: minimizing the makespan of a task graph. In [120], Yu et al. consider the scheduling problem in a "pay-per-use" service Grid. Their objective is to minimize the total cost for executing a workflow on such a Grid, and at the same time, QoS which is interpreted as a deadline, is provided. The algorithm firstly partitions the original DAG into sub-workflows, which consist of a sequential set of tasks between two synchronization tasks (the nodes from which at least one sub-workflow starts and/or ends) in the graph, and assigns a sub-deadline to each partition by a combinational Breadth-First Search and Depth-First Search with critical path analysis. Then for each partition a planning process is applied to find the optimal mapping for which the cost is the lowest and the deadline is met. The optimal search is modeled by a Markov Decision Process and is implemented by using a dynamic programming algorithm. Rescheduling is also provided in this work when a sub-workflow misses its sub-deadline. But only the sub-workflow's children will be rescheduled to reduce the rescheduling cost. A similar problem is also considered by Sample et al. in [92]. They adopt the similar scheduling procedures as Yu's. The scheduling begins with the selection of an initial schedule based on service providers' estimates for completion time and fee. If the certainty of the completion time and cost is dropped to a threshold, which is usually caused by performance fluctuation, rescheduling will be carried out. To find an initial scheduling, the scheduler requests bids from resource providers for the services they can provide. The bid request is based on the service needed, the expected start time for the service, and information about the size and complexity of the input parameters to the service. The scheduler's decision as to which bid is finally accepted is based on the *Pareto optimality* of the best schedules. For a bid to be Pareto optimal there can be no other bid with an absolute advantage in terms of price, time and certainty of provided performance.

The uncertainty introduced by the dynamism also brings opportunities for the application of another method in the Grid: the data-and-control dependency task graph (DCG) scheduling [41]. Different to a DAG, a DCG has two types of edges denoting data dependency and control dependency among task nodes respectively. A control-dependency edge usually represents some condition relying on the results of its starting node, so in a DCG, users can predefine adaptive rules for dynamic adaptation in their task graph. For example, a rule can be duplicating all the successors of a node if its real execution time fails to match its predicted one. Research in [41] gives an example of how to schedule such conditional task graphs in terms of reducing the final possible makespan. However, no research that adopts this idea to the Grid environment exists at this time.

3.5 Data Scheduling

In high energy physics, bioinformatics, and other disciplines, there are applications involving numerous, parallel tasks that both access and generate large data sets, sometimes in the petabyte range. Data sets in this scale require specialized storage and management systems and data Grid projects are carried out to harness geographically distributed resources for such data-intensive problems by providing remote data set storage, access management, replication services [15], and data transfer protocols [5].

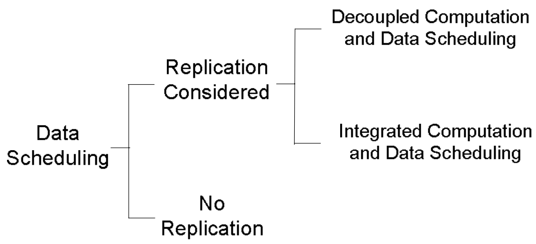

Fig. 10: Taxonomy of algorithms considering data scheduling.

As mentioned in Section 1, one important difference between Grid scheduling and its traditional counterpart is that, in the latter, the data staging problem usually need not be considered by scheduling algorithms. This is because the resources on which applications will run are determined before scheduling so that the data staging cost is a constant. The only cost relating to data transmission that should be considered is from the data produced at the run time, e.g., the data dependency in DAGs. In the Grid environment, by contrast, the location where an application finally is processed is usually selected in real time, so that the cost to transfer the input data from the storage sites to the processing sites might vary according to which processing sites are selected as well as which storage sites are used when the data have multiple replicas. Further, assigning a task to the machine that gives its best execution time may result in poor performance due to the cost of retrieving the required input data from data repositories. In [82], Park et al classify models of cost measured in makespan into five categories, namely 1) Local Data and Local Execution (here, local means where a job is submitted), 2) Local Data and Remote Execution, 3) Remote Data and Local Execution, 4) Remote Data and Same Remote Execution and 5) Remote Data and Different Remote Execution. The algorithms we have discussed in previous subsections can not be directly used to solve these problems, although they are helpful to find feasible answers. In Fig. 10, the taxonomy of algorithms which consider the data scheduling problem is given.

- **On Data Replication**

When the scheduling problem with data movement is considered, there are two situations: whether data replication is allowed or not. In Pegasus, the CWG assumes that accessing an existing dataset is always more preferable than generating a new one [36] when it maps an abstract workflow to a concrete one. By contrast, Ranganathan et al [85] view data sets in the Grid as a tiered system and use dynamic replication strategies to improve data access. These data replication strategies work at the system level instead of the application level, aiming to reduce the bandwidth consumption for data transfer and the workload of data hotspots.

- **On Computation and Data Scheduling**

When the interaction of computation scheduling and data scheduling is considered, we can also find two different kinds of approaches: decoupling computation from data scheduling [86] or conducting a combinational scheduling [4].

Ranganathan et al. [86] consider the dynamic job scheduling with data staging requirement. Data replication is provided to reduce the communication bandwidth consumption and avoid data access hotspots. Computation scheduling and data replication strategies are independent from each other. In [86], four simple dynamic computation scheduling algorithms (namely, *JobRandom* which selects a random site for computation; *JobLeastLoaded* which selects the least loaded site; *JobDataPresent* which always selects a site where the required data is already there, and *JobLocal* which always runs a job at the local site) and three dada scheduling algorithms (namely, *DataDoNothing* in which no active replication takes place; *DataRadom* which replicates a popular data set to a random site, and *DataLeastLoaded* which selects the least loaded sites as the destination for a new data set replication). So there are in total 4*3 = 12 scheduling algorithms. Without tight coupling with requirements of applications, it is obvious that these algorithms are for system-centric purposes.

Actually, which resources are used to run the tasks and where the data are accessed are two inter-related issues because the choice of the execution site may depend on the location of the data and the data site selection may depend on where the computation will take place. If the data sets involved in the computation are large, one may favor moving the computation close to the data. On the other hand if the computation is significant compared to the cost of data transfer the compute site selection could be considered first. Based on this idea, Alhusaini et al [4] combine scheduling of the computation with data replica selection to reduce the makespan of a collection of applications which can be unified to compose a DAG. The cost to access data repositories and data transfer are accounted when computing the total time cost of a task in a DAG. A static level-by-level algorithm is used to search for an optimal scheduling. In this algorithm, the original DAG is partitioned along with the directed edges into levels and in the same level, all tasks are independent. Min-min and Max-min are applied to schedule tasks in the same level. But, how to find the levels when the task graph is unbalanced is not mentioned in this paper, and as discussed earlier, a level-by-level partition can not harness locality to facilitate tasks having dependency.

In [39], Desprez et al try to optimize the throughput of a Grid system for independent jobs each of which requests to refer to a certain data set, under the constraint of storage capacity of each computation site. The algorithm is based on the following assumptions: data replication is allowed; a job can only refer to local data on the same site whose storage capacity is limited; the execution time of a job is linear with the size of the data set to which it refers; the number of each kind of jobs in the job set follows a fixed proportion. The objective is to maximize the total size of the job set. An integer solution to a linear program problem is used as an approximation, because the problem itself is NP Complete. By simulation, the authors also verified their method by comparison against the MCT and Max-Min algorithm.

In [109], Venugopal et al propose a scheduling algorithm considering two kinds of cost: the economic budget and time. The algorithm tries to optimize one of them under a limited number of the other, e.g., spend a budget as small as possible, while not missing the deadline. The incoming applications consist of a set of independent tasks each of which requires a computational resource and accesses a number of data sets located on different storage sites. The algorithm assumes every data set has only one copy in the Grid, so that the resource selection is only for computational resources, taking into account the communication costs from data storage sites to different computation sites as well as computation costs. Instead of doing a thorough search in a space whose size is exponential in the number of data sets requested by a task, the resource selecting procedure simply performs a local search which only guarantees that the current mapping is better than the previous one. In this way, the cost of the search procedure is linear. But the drawback of this strategy is that it is not guaranteed to find a feasible schedule even if there is one.

3.6 Non-traditional Approaches for Grid Task Scheduling

The Grid is a kind of system made up of a large number of autonomous resource providers and consumers, which are running concurrently, changing dynamically, interacting with each other but self-ruling. In nature and human society, there are some systems having the similar characteristics. Therefore new approaches, such as the Grid Economy [20] and other heuristics inspired by natural phenomena, were proposed in recent years to address the challenges of Grid computing. Ideas behind these approaches are not originally applied to the scheduling problem and mapping from the problem spaces in which they are initially used to Grid scheduling problems is usually required. Fig. 11 depicts the framework of our discussion in this subsection.

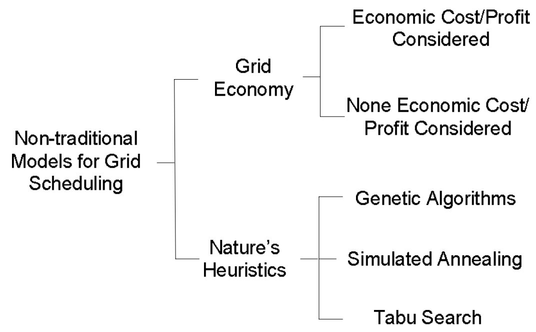

Fig. 11: Taxonomy of non-traditional scheduling approaches.

3.6.1 Grid Economic Model

If the features of Grid computing are inspected carefully and compared with other systems around us, it can be found that the Grid has high similarity with the market. The market is also a decentralized, competitive and dynamic system where consumers and product providers have their own objectives. Based on this observation, economic models are introduced to the Grid in order to optimize the resource management and scheduling problems [19]. The basic components in a market are producers, consumers, and commodities, analogous to resource owners, resource users (applications), and various computing resources in the Grid environment. Economic theories are established based on the study of market behaviors. Price and quality of commodities are parameters to the decision making procedures of consumers and providers. For example, a consumer usually wants to get better services (e.g., a smaller makespan) with a budget as small as possible, while a provider usually wants to get a higher resource utilization to raise its profits.

The use of economic methods in Grid scheduling involves interacting processes between resource providers and users, analogous to various market behaviors, such as bargain, bid, auction and so on. Buyya et al [19] discuss some economic models that can be applied to the Grid world, including the Commodity Market Model, Tender/contract-net Model, and Auction Model.

As economic models are introduced into Grid computing, new research opportunities arise. Because the economic cost and profit are considered by Grid users and resource providers respectively, new objective functions and scheduling algorithms optimizing them are proposed. In [18], several algorithms called deadline and budget constrained (DBC) scheduling algorithms are presented which consider the cost and makespan of a job simultaneously. These algorithms implement different strategies. For example, guarantee the deadline and minimize the cost or guarantee the budget and minimize the completion time. The difficulties to optimize these two parameters in an algorithm lie in the fact that the units for cost and time are different, and these two goals usually have conflicts (for example, high performance resources are usually expensive). We can also find such examples in [120] and [109] discussed earlier.

Economic models are not only useful when economic cost/profit are explicitly involved, their basic ideas are also useful to construct new scheduling methods with traditional objectives. For example in [43], the tender model is used to schedule Grid applications. The applications' goal is to minimize their makespan by selecting those resources that can complete them the earliest. However, the resource providers' goal is to insert the incoming tasks into their spare time slots in order to maximize the resource utilization. In this model, both consumers and providers are autonomous (have their own objectives), so economic models are very adaptive to the autonomy and diversity both in Grid users and resources. But in [43], only the computational resources are considered as competitive resources, so the model can be improved by incorporating network links with computational resources. In [124], a similar idea is applied, where Grid jobs' priorities are considered and related to the budgets the jobs would like to spend. In [119], the Grid scheduling problem is modeled using game theory, which is a technique commonly used to solve economic problems. Each task is modelled as a player whose available strategies are the resources on which the task could run. The payoff for a player is defined to be the sum of the benefit value for running the task and all communication arriving at that task. Game theory examines the existence of a stable solution in which players are unable to improve their payoff by changing their strategy. The solution found is not always optimal for the collective payoff, but in many cases it is close to optimal.

Economic methods for scheduling problems are very interesting because of their successes in our daily lives. The models mentioned above can only support relatively simple Grid scheduling problems such as independent tasks. For more complex applications, such as those consisting of dependent tasks and requiring cross-site cooperation, more sophisticated economic models might be needed.

3.6.2 Scheduling Methods Inspired by Nature's Laws

As scheduling is usually a process to find optimal solutions, several analogies from natural phenomena have been introduced to form powerful heuristics, which have proven to be highly successful. Some of the common characteristics of Nature's heuristics are the close resemblance to a phenomenon existing in nature, namely, non-determinism, the implicit presence of a parallel structure and adaptability [1]. Abraham et al [1] and Braun et al [17] present three basic heuristics implied by Nature for Grid scheduling, namely, *Genetic Algorithm* (GA), *Simulated Annealing* (SA) and *Tabu Search* (TS), and heuristics derived by a combination of these three algorithms.

- **Genetic Algorithm (GA)**

GA is an evolutionary technique for large space search. The general procedure of GA search is as follows [17]:

1) Population generation: A population is a set of *chromosomes* and each represents a possible solution, which is a mapping sequence between tasks and machines. The initial population can be generated by other heuristic algorithms, such as Min-min (called seeding the population with a Min-min chromosome).
2) Chromosome evaluation: Each chromosome is associated with a fitness value, which is the makespan of the task-machine mapping this chromosome represents. The goal of GA search is to find the chromosome with optimal fitness value.
3) Crossover and Mutation operation: Crossover operation selects a random pair of chromosomes and chooses a random point in the first chromosome. For the sections of both chromosomes from that point to the end of each chromosome, crossover exchanges machine assignments between corresponding tasks. Mutation randomly selects a chromosome, then randomly selects a task within the chromosome, and randomly reassigns it to a new machine.
4) Finally, the chromosomes from this modified population are evaluated again. This completes one iteration of the GA. The GA stops when a predefined number of evolutions is reached or all chromosomes converge to the same mapping.

This genetic algorithm randomly selects chromosomes. Crossover is the process of swapping certain sub-sequences in the selected chromosomes. Mutation is the process of replacing certain sub-sequences with some task-mapping choices new to the current population. Both crossover and mutation are done randomly. After crossover and mutation, a new population is generated. Then this new population is evaluated, and the process starts over again until some stopping criteria are met. The stopping criteria can be, for example, 1) no improvement in recent evaluations; 2) all chromosomes converge to the same mapping; 3) a cost bound is met.

For its simplicity, GA is the most popular Nature's heuristic used in algorithms for optimization problems. In the realm of Grid scheduling, we can find other examples in [66], [3], [99] [98] and [110].

- **Simulated Annealing (SA)**

SA is a search technique based on the physical process of annealing, which is the thermal process of obtaining low-energy crystalline states of a solid. At the beginning, the *temperature* is increased to melt the solid. If the temperature is slowly decreased, particles of the melted solid arrange themselves locally, in a stable "ground" state of a solid. SA theory states that if temperature is lowered sufficiently slowly, the solid will reach thermal equilibrium, which is an optimal state. By analogy, the thermal equilibrium is an optimal task-machine mapping (optimization goal), the temperature is the total completion time of a mapping (cost function), and the change of temperature is the process of mapping change. If the next temperature is higher, which means a worse mapping, the next state is accepted with certain probability. This is because the acceptance of some "worse" states provides a way to escape local optimality which occurs often in local search [73].

A SA algorithm is implemented in [17]. The initial system temperature is the makespan of the initial (random) mapping. The initial SA procedure implemented here is as follows. The first mapping is generated from a uniform random distribution. The mapping is mutated in the same manner as the GA, and the new makespan is evaluated. If the new makespan is better, the new mapping replaces the old one. If the new makespan is worse (larger), a uniform random number $z \in [0, 1)$ is selected. Then, z is compared with y, where

$$y = \frac{1}{1 + e^{\left(\frac{\text{old makespan-new makespan}}{\text{temperature}}\right)}}$$

If $z > y$, the new (poorer) mapping is accepted; otherwise it is rejected, and the old mapping is kept. So, as the system temperature "cools", it is more difficult for poorer solutions to be accepted. This is reasonable because when the temperature is lower, there is less possibility to find a better solution starting from another poorer one. After each mutation, the system temperature is reduced to 90% of its current value. (This percentage is defined as the cooling rate.) This completes one iteration of SA. The heuristic stops when there is no change in the makespan for a certain number of iterations or the system temperature approaches zero. Another example of SA's application in the Grid can be found in [119].

- **Tabu Search (TS)**

TS is a meta-strategy for guiding known heuristics to overcome local optimality and has now become an established optimization approach that is rapidly spreading to many new fields. The method can be viewed as an iterative technique which explores a set of problem solutions, denoted by X, by repeatedly making moves from one solution s to another solution s' located in the neighbourhood $N(s)$ of s. These moves are performed with the aim of efficiently reaching an optimal solution by minimizing some objective function $f(s)$.

In [17], a TS is implemented beginning with a random mapping as the initial solution, generated from a uniform distribution. To manipulate the current solution and move through the solution space, a *short hop* is performed. The intuitive purpose of a short hop is to find the nearest local minimum solution within the solution space. The basic procedure to perform a short hop is to consider, for each possible pair of tasks, each possible pair of machine assignments, while the other assignments are unchanged. If the new makespan is an improvement, the new solution is saved, replacing the current solution. The short hop procedure ends when (1) every pair-wise remapping combination has been exhausted with no improvement, or (2) the limit on the total number of successful hops ($limit_{hops}$) is reached. When the short hop procedure ends, the final mapping from the local solution space search is added to the tabu list. The tabu list is a method of keeping track of the regions of the solution space that have already been searched. Next, a new random mapping is generated, and it must differ from each mapping in the tabu list by at least half of the machine assignments (a successful *long hop*). The intuitive purpose of a long hop is to move to a new region of the solution space that has not already been searched. After each successful long hop, the short hop procedure is repeated. The stopping criterion for the entire heuristic is when the sum of the total number of successful short hops and successful long hops equals $limit_{hops}$. Then, the best mapping from the tabu list is the final answer.

- **Combined Heuristics**

GA can be combined with SA and TS to create combinational heuristics. For example, The Genetic Simulated Annealing (GSA) [1] heuristic is a combination of the GA and SA techniques. In general, GSA follows procedures similar to the GA outlined above. However, for the selection process, GSA uses the SA cooling schedule and system temperature and a simplified SA decision process for accepting or rejecting a new chromosome.

These Nature's heuristics were only relatively introduced into the scheduling area and more work needs to be done to fit them in a Grid context. There are a lot of interesting questions [73]. First, the meaning of controlling measurements in such heuristics needs to be refined. For example, in SA, the cooling rate is usually set to a fixed value to control the number of iterations the search process will perform; the question is: how to choose this value as well as the threshold value *y* for accepting a poorer mutation? Second, there is a trade-off between the search cost and the degree of optimality of solutions found. For example, in a genetic algorithm, historical knowledge can be used to guide the chromosome selection, crossover, or mutation process so that the search process can converge quickly. But this adjustment seems to contradict the philosophy of an evolutionary algorithm: randomization and diversity generate better results, and it may not bring a better solution. Third, what is the effect of the dynamic nature of the Grid on these algorithms? More practical experiments should be done to find proper answers to these questions.

3.7 Scheduling under QoS Constraints

In a distributed heterogeneous non-dedicated environment, quality of services (QoS) is a big concern of many applications. The meaning of QoS can be varied according to the concerns of different users. It could be a requirement on the CPU speed, memory size, bandwidth, software version or deadline. In general, QoS is not the ultimate objective of an application, but a set of conditions to run the application successfully.

Two methods of dealing with the heterogeneity and variability have been proposed [40]. The *guaranteed approach* is based on the notion of negotiating and re-negotiating quality of service contracts with the resource providers. These QoS management infrastructures assume that some form of service guarantees can be provided by resource providers. An example of this approach can be found in [25]. The *best effort approach* assumes that guarantees cannot be obtained from the system. In these adaptive schemes, algorithms that enable the applications to adapt themselves to the operating environments are built into the application. Adaptive examples are demonstrated in [46] and [80].

However, the concerns of current research on QoS, including the examples above, are at the resource management level rather than at the task/resource scheduling level. Once QoS is guaranteed by resources or adjusted by users, the rest of the scheduling process falls back into the traditional mode. Being aware of this limitation, He et al give the first study in [57], which attempts to analyze QoS in Grid task scheduling. QoS information is embedded into the Min-min scheduling algorithm to make a better match among different levels of QoS request/supply. Tasks can only be mapped to resources which are qualified for the QoS, and tasks with higher QoS requirements are mapped before those with lower requirements. Min-min is used to determine the order of mapping among tasks having the same QoS level. As the first effort, the work in [57] certainly leaves many open issues: only one dimensional QoS (network bandwidth) is considered, and tasks are assumed independent. How does the imposition of QoS on a few tasks affect the scheduling of other tasks in other application models such as DAG, and how to optimize other objectives besides QoS are interesting research topics.

3.8 Strategies Treating Dynamic Resource Performance

As discussed, one of the most significant difficulties for task scheduling in the Grid is the dynamic nature of Grid resources. The dynamic changing of resource performance mainly comes from the autonomy of the sites, and the contention incurred by resource sharing among a potentially large number of users. Current Grid scheduling systems adopt three levels of techniques to relieve the dynamic problem: scheduling based on just-in-time information from Grid information services (e.g., dynamic algorithms for load balancing), performance prediction, and dynamic rescheduling at run time, as Fig. 12 shows.

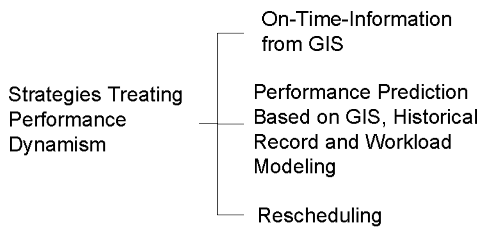

Fig. 12: Taxonomy of scheduling strategies treating performance dynamism.

- **Grid Information Service (GIS)**

A GIS is a software component, singular or distributed, that maintains information about people, software, services and hardware that participate in a computational Grid, and makes that information available upon request. For a Grid scheduler, the most important functionalities a GIS can provide are resource discovery and monitoring in a dynamic way. Resource discovery can tell a scheduler what new resources are available and where they are, and resource monitoring can provide important up-to-date information about resource and data set location status, upon which scheduling decisions might be based. The Grid information service itself is an important research topic in the computational Grid and is intensively studied. Aktaruzzaman gives a comprehensive review about this topic in his survey [7]. The most popular GIS used by current Grid problems include the Metacomputing Directory Service (MDS [33], also denoted as Monitoring and Discovery Service) [33], the Network Weather Service (NWS) [111], Ganglia [89], Giggle [29] etc.

- **Performance Prediction**

In dynamic systems, predictions are important for decision making. Good examples of existing predictions are weather forecasting and stock market analysis. In terms of task scheduling, almost all static algorithms rely on performance estimates when conducting the scheduling and we have seen a lot of such examples above. It is usually easy to get a resource's static information, such as CPU frequency, memory size, network bandwidth, file systems, etc. But run-time information, such as CPU load, available memory, and available network bandwidth, is comparatively hard to obtain due to the performance fluctuation brought by contention among sharing resources.

- *On Prediction Accuracy*

When a scheduling decision is made, an accurate prediction on run-time resource measurement parameters is critical to achieve the ultimate objective. For example, Takefusa et al [106] present a deadline scheduling algorithm for client-server systems in the Grid. The estimated service processing time is computed with a parameter decided by the scheduler's confidence for the prediction accuracy. If the performance prediction is wrong, resource utilization will drop or services will fail due to missed deadlines. Schopf et al [93] use stochastic values to represent variable behaviors of resources which can be obtained from NWS, and assume these values can be adequately represented by using normal distributions. Based on these assumptions, they define a *time-balancing* scheduling strategy for data-parallel applications. The amount of data distributed to each resource is based on the standard deviation of the predicted completion time. Another example using prediction accuracy as a parameter during scheduling can be found in [65].

- *Prediction Based on Historical Record*

According to how the historical data on previous resource availability information and job performance records is utilized and what dynamic parameters are considered, predictions can be made in a simple or a complex manner. A comprehensive survey on this issue is another theme, but Yang et al [116] provide a very good example. In [116] three levels of prediction are presented. The first level is the simplest one-step-ahead prediction which only considers the tendency of the CPU load changing and has historical peak values as thresholds. If the performance of current time point is increasing, the performance for the next time point will be predicted as increasing unless the upper threshold is reached, and vice versa. Based on this one point prediction, interval performance prediction for the near future can be obtained by aggregating multiple point performance in previous intervals (for example by taking the average value) and using the one-step-ahead prediction scheme to the aggregated value. The variance of performance in an interval can also be obtained in a similar way: using historical data of a series points to get the standard deviation of the performance during each interval, and then applying the one-step-ahead scheduling scheme to standard deviations of previous intervals. A machine with lower interval variance is considered more "reliable", and a scheduler "conservatively" assigns less work to high-variance resources.

- *Prediction Based on Workload Modeling*

In the Grid, resource performance change is usually caused by contention among remote Grid applications and local applications of resources. Another idea for performance prediction is based on this observation: if the arrival of applications can be modeled, performance prediction could be improved. For example, in [56], He et al assume that the background workload coming to mulitclusters (where Grid applications are running) is a static linear function of time. Gong et al [54] study a more complex case. They propose a performance model of parallel tasks on non-dedicated heterogeneous resources where resource owners' local jobs will preempt remote Grid tasks. They assume that the arrival of local jobs will follow a Poisson distribution, and the probability function for the completion time of parallel remote jobs is given based on this assumption. This paper does not provide a material prediction service but a prediction formula which gives an upper bound and a measure of feasibility for parallel processing in a non-dedicated distributed environment. In [105], Sun et al apply the results in [54] in a Grid performance prediction and scheduling system called Grid Harvest. Performance of available resources is predicted by the performance model in [54], which takes measured parameters as input. The parameters include the local job arrival rate on each machine, the machine utilization, the standard deviation of service time, and the computing capacity of each machine. The parameters can be obtained by running standard benchmarks, and exploring the historical records. The limitation of the performance model and prediction method in [54] and [105] is that they are only applicable to divisible independent jobs, not to DAGs; besides, they only consider the workload but not the communication and synchronization. Thus the application of this model is restricted.

- **Rescheduling**

When there is no resource prediction service available or the resource prediction can not provide an accurate forecast, rescheduling which changes previous schedule decisions based on a fresh resource status, can be taken as a remedy. With the efforts of system designers and developers, more and more Grid infrastructures now support job migration, which is one precondition of rescheduling [69] [108]. Algorithms taking rescheduling into consideration have been mentioned in previous discussions (refer 3.3 and 3.4.2), and here, some details will be given by examples.

GrADS [31] has introduced the concept of a performance contract: an agreement between a user and the provider of one or more resources. A contract states that when provided with certain resources with certain capabilities and certain problem parameters, the application will achieve a specified sustained level of measurable performance. When a contract violation is detected, a rescheduler will be activated. The rescheduling process must determine whether a rescheduling is profitable, based on sensor data, estimates of the remaining work in the application, and cost of moving to new resources. If the rescheduling appears profitable, the rescheduler will compute a new schedule and contact *rescheduling actuators* located on each processor. These actuators use some mechanisms to initiate the actual migration. Two rescheduling mechanisms are used in GrADs: rescheduling by stop and restart (RSR) and rescheduling by processor swapping (RPS).

In the RSR approach, an application is suspended and migrated only when better resources are found. When a running application is signaled to migrate, all application processes check user specific data and terminate. The rescheduled execution is then launched by restarting the application on the new set of resources, which then read the checkpoints and continue the execution. Although this approach is flexible, in practice, it could be very expensive: each migration event can involve large data transfers. Moreover, restarting the application can incur expensive start-up costs, and significant application modifications may be required for a specialized restart code. Therefore, the RPS approach is introduced. The basic idea of RPS is similar to the duplication heuristics. Application is launched to more machines than the number that will actually be used for the computation. Some of these machines become a part of the computation (the *active* set), while some others do nothing initially (the *inactive* set). The user's application can only see the active processes. During execution, the contract monitor periodically checks the performance of the machines and swaps slower machines in the active set with faster machines in the inactive set. This approach requires little application modification and provides an inexpensive fix for many performance problems. However, the approach is less flexible than RSR – the processor pool is limited to the original set of machines, and the data allocation can not be modified.

In [114], a self-adaptive scheduling algorithm is given to reschedule tasks on the processors showing "abnormal" performance. "Abnormal" performance means that the behaviors of those processors violate the performance model proposed in [105] and [54] with an unexpected high local jobs arrival rate that delays remote Grid jobs. The algorithm uses a prediction error threshold to trigger the rescheduling process. If the estimated complete time of a Grid task is shortened after migration, the tasks will be migrated to the processor which can give it minimum complete time according to current prediction. The key problem in this algorithm is to find a proper threshold to determine whether a processor is abnormal. In this paper, the effect of different thresholds is tested by simulations, but it could vary if different settings are given in practice. Another problem with this algorithm is that it ignores the migration cost when it computes the benefits of migrations.

The above methods only consider the rescheduling of independent tasks in the Grid. A novel low cost rescheduling policy is proposed in [90] to improve the initial static schedule of a DAG. This algorithm only considers a selective set of tasks for rescheduling based on measurable properties. The key idea of this selective rescheduling policy is to evaluate (at run-time, before each task starts execution) the starting time of each node against its estimated starting time in the static scheduling and the maximum allowable delay, in order to make a decision for rescheduling. The maximum allowable delay could be the *slack* of a task, or the *minimal spare time*. The slack of a task is defined as the maximal value that can be added to the execution time of a task without affecting the overall makespan of the schedule. The minimal spare time is defined as the maximum value that can be added to the execution time of a task without delaying its successors' execution. As the tasks of the DAG are running, the rescheduler maintains two schedules. One is for the static scheduling using estimated values, and the other for keeping track of the status of the tasks executed so that the gap between the original schedule and the real run can be known. Once the gap is beyond the slack or the minimal spare time, rescheduling to all unexecuted tasks will be triggered and a new slack or minimal spare time for each task is recomputed. This rescheduling policy is applied to some list algorithms discussed in 3.4.2 such as HEFT, FCP and Hybrid. Simulation results show that this policy can achieve comparable performance with the policy that reschedules all tasks in the DAG while dramatically reducing the rescheduling overhead.

4. Summary and Open Issues

In this literature review, the task scheduling problem in Grid computing is discussed, mainly from the aspect of scheduling algorithms. Although task scheduling in parallel and distributed systems has been intensively studied, new challenges in Grid environments still make it an interesting topic, and many research projects are underway. Through our survey on current scheduling algorithms working in the Grid computing scenario, we can find that heterogeneity, dynamism, computation and data separation are the primary challenges concerned by current research on this topic. We also find that the evolution of Grid infrastructures, e.g., supports for complex application models such as DAG, resource information services and job migration frameworks, provides an opportunity to implement sophisticated scheduling algorithms. In addition to enhancements to classic scheduling algorithms, new methodologies are applied, such as the adaptive application-level scheduling, Grid economic models and Nature's heuristics.

Due to the characteristics of Grid computing and the complex nature of the scheduling problem itself, there are still many open issues for Grid scheduling [8], relating to scheduling architectures, protocols, models etc. However, we only focus on those questions about algorithms. Eschewing any specific assumptions, we find the following general issues deserving of future exploration.

- **Application and Enhancement of Classic Heterogeneous Scheduling Algorithms in Grid Environments** [1]

We have seen that many solutions for Grid scheduling are inspired by classic algorithms for traditional systems. But the potentials of these algorithms in the Grid scenarios haven't been fully exploited. For example, in 3.4.2, we disused mainly three kinds of heuristics used for scheduling of tasks with precedence orders in heterogeneous parallel and distributed systems: list heuristics, duplicated heuristics and clustering heuristics. These heuristics were first invented for DAG scheduling in homogeneous systems, and then amended for heterogeneous systems. Although the ideas behind the latter two categories also have many advantages in the Grid scenario, almost all algorithms in the current literature refer to list algorithms. Since all of these heuristics consider complex application models, where tasks can be fine granular and with data and control dependency, there is great potential for using these heuristics in Grid computing as the underlying supporting infrastructures continue to develop.

- **New Algorithms Utilizing Dynamic Performance Prediction** [2]

The dynamism in the Grid jeopardizes the assumptions on which class approximation algorithms optimizing makespans of applications stand, as well as static performance estimates used by most of static heuristics. To deal with performance variation in the Grid, resource information and prediction are introduced in many Grid projects. As the techniques in this field develop, better performance knowledge prior to the task scheduling stage can be expected. But current scheduling algorithms only consider a snapshot value of the prediction when they make the estimate, and assume that value is static during the job execution period. This might be a waste of the prediction efforts which can actually provide continuous variation information about the system. So heuristics that can exploit multiple stage prediction information should be designed. Compared to current models of heterogeneous heuristics, the resource models for such algorithms should be remedied with an extra matrix to present the performance values as time proceeds. Fig. 13 shows an example. It can be expected that these heuristics will be more complex than current ones

[1] Recall 3.4.1and 3.4.2
[2] Recall 3.4.1and 3.4.2

that only consider the heterogeneity among resources rather than the variation of performance. Another question we can ask is whether we can reestablish approximating for makespan optimization based on performance predictions. For example, if we know the range of performance fluctuation is bounded, can we find a bound for the ratio of real makespan to optimal finish time accordingly?

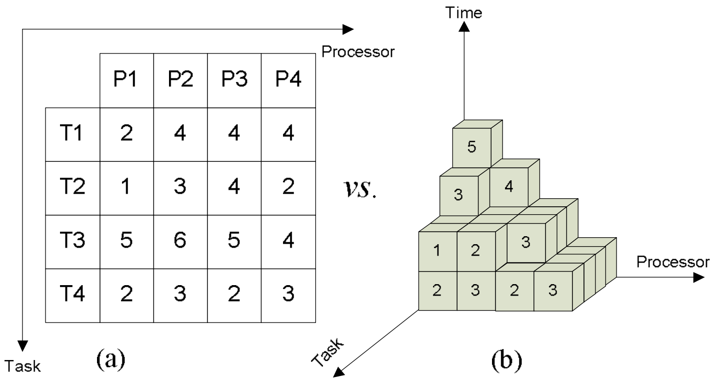

Fig. 13: (a) A 2-D matrix depicts predictions on processor performance in traditional systems. (b) In Grid computing, performance prediction has a 3-D structure.

- **New Rescheduling Algorithms Adaptive to Performance Variation** [3]

As discussed in 3.8, the problem with current rescheduling algorithms in the Grid is high cost and lack of consideration of dependent tasks. For jobs whose makespans are large, rescheduling for the original static decisions can improve the performance dramatically. However, rescheduling is usually costly, especially in DAGs where there are extra data dependencies among tasks compared to independent applications. In addition, many other problems also exist, for example when the rescheduling mechanisms should be invoked, what measurable parameters should decide whether a rescheduling is profitable, and where tasks should be migrated. Current research on DAG rescheduling leaves a wide open field for future work.

- **New Algorithms under QoS Constraints** [4]

QoS is the concern of many Grid applications. Most current research concentrates on how to guarantee the QoS requirements of the applications like [60], but few of them study how the QoS requirements affect the resources assignment and then the performance of the other parts of the applications. For example, in Fig. 14, tasks 2, 3, and 6 have QoS requirements which can only be satisfied by a certain resource, so that they can only be assigned to that resource. Then, how should the remaining part of the DAG be scheduled under the constraint that a certain resource will be definitely occupied by some tasks for a certain time? Situations might become more complex when there are more tasks having QoS requirements and more optional resources.

[3] Recall 3.8
[4] Recall 3.7.

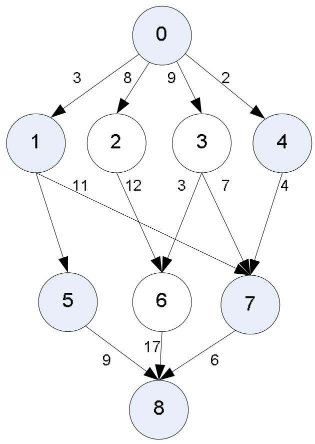

Fig. 14: An example of DAG with nodes having QoS requirements.

- **New Algorithms Considering Combined Computation and Data Scheduling**[5]

As discussed, scheduling algorithms in traditional computing paradigms barely consider the data transfer problem during mapping computational tasks, and this neglect will be costly in the Grid scenario. Only a handful of current research efforts consider the simultaneous optimization of computation and data transfer scheduling, which brings opportunities for future studies. Fig. 15 describes a simple example showing how taking data staging into account will effect the scheduling. If data staging is not considered, the critical path of DAG in Fig. 15 (a) is $T1$-$T2$-$T4$, but if data transfer cost is considered, the critical path will change to $T1$-$T3$-$T4$. This consideration also has effect on the selection of computational node for a task: if no data transfer cost, $T2$ can complete earlier on $P1$, but if there is such a cost, $P2$ is preferred. The situation could be far more complex if there are multiple copies of $D1$ and $D2$, and data dependencies among T_x are considered.

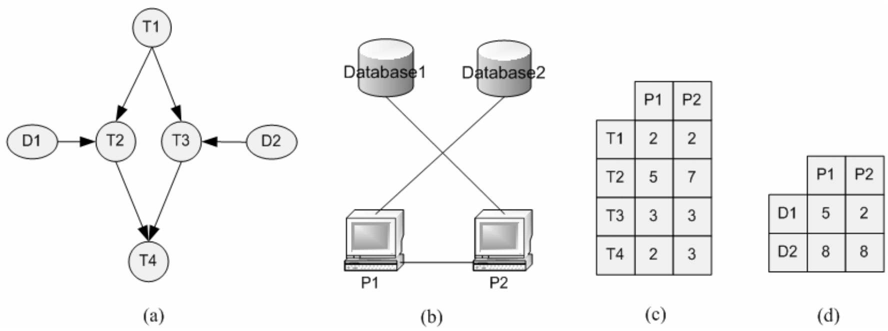

Fig. 15: (a) A DAG with input data D1 and D2. Suppose data dependency between any two computation nodes T_x and T_y is 0. (b) Resources available for the job in (a); P1 and P2 are two computational nodes; D1 is on Database1 and D2 is on Database2. (c) Computational cost estimate. (d) Data Transfer cost estimate.

[5] Recall 3.5

- **New Problems Introduced by New Models** [6]

As new methodologies are introduced, new problems arise, in connection with the choice of objective functions and the mapping of a classic scheduling problem to the new model. For example, new objective functions considering the economic cost of computing and their optimization are far from having been thoroughly studied. As well the economic models in Grid computing themselves need to be refined to apply to more sophisticated applications which include task dependency and require cooperation among different resource providers. And in a genetic scheduling algorithm, one open problem is how to use dynamically predicted performance information to help the speed of convergence (as the current genetic scheduling algorithms are static).

- **New Algorithms Utilizing the Grid Resource Overlay Structure**

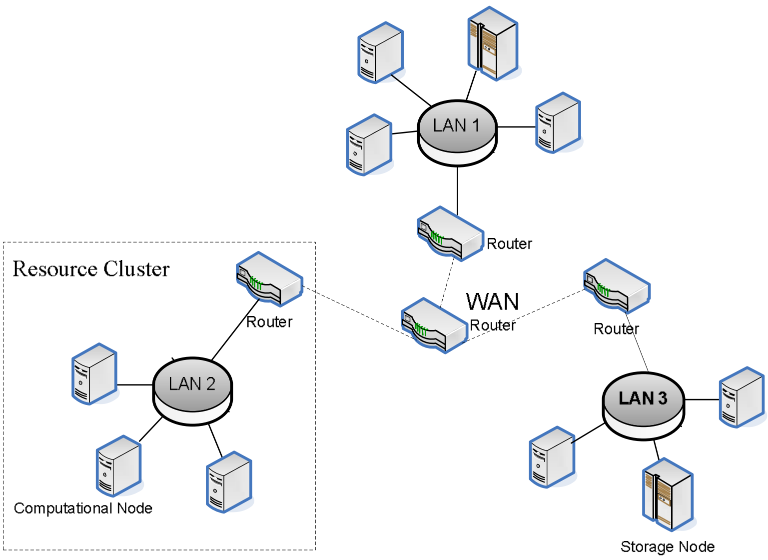

Fig. 16: Clustered distribution of Grid resources.

Although the Grid has the characteristics of heterogeneity and dynamicity, these features are not flatly distributed in resources, but are rather distributed hierarchically and locally in many cases, due to the composition of the Grid resources. Current Grid resources are usually distributed in a clustered fashion. Fig. 16 depicts an example of clustering distributed Grid resources. Resources in the same cluster usually belong to the same organization and are relatively more homogeneous and less dynamic in a given period. Inside a cluster, communication cost is usually low and the number of applications running at the same time is usually small. These distribution properties might bring another possibility for new algorithms to deal with the Grid challenges. For example, by taking multiphase or multilevel strategies, a Grid scheduler can first find a coarse scheduling in the global Grid and then a fine schedule in a local cluster. This type of strategy has the following advantages:

[6] Recall 3.6.

- At the higher level, where fine resource information is harder to obtain, the global scheduling can use coarse information (such as load balancing, communication delay of WAN links) to provide decentralized load balancing mechanisms.
- At the lower level, it is easy for local scheduling to utilize more specific information (such as information from a local forecaster) to make adaptive decisions.

References

[1] A. Abraham, R. Buyya and B. Nath, Nature's Heuristics for Scheduling Jobs on Computational Grids, in Proc. of 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), pp. 45-52, Cochin, India, December 2000.

[2] A. K. Aggarwal and R. D. Kent, *An Adaptive Generalized Scheduler for Grid Applications*, in Proc. of the 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'05), pp.15-18, Guelph, Ontario Canada, May 2005.

[3] M. Aggarwal, R.D. Kent and A. Ngom, *Genetic Algorithm Based Scheduler for Computational Grids*, in Proc. of the 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'05), pp.209-215, Guelph, Ontario Canada, May 2005.

[4] A.H. Alhusaini, V.K. Prasanna, and C.S. Raghavendra, *A Unified Resource Scheduling Framework for Heterogeneous Computing Environments*, in Proc. of the 8th Heterogeneous Computing Workshop, pp.156-165, San Juan, Puerto Rico, April 1999.

[5] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, *The Globus Striped GridFTP Framework and Server*, in Proc. of the 2005 ACM/IEEE conference on Supercomputing, pp.54-64, Seattle, Washington USA, November 2005.

[6] M, Arora, S.K. Das, R. Biswas, *A Decentralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments*, in Proc. of International Conference on Parallel Processing Workshops (ICPPW'02), pp.:499 – 505, Vancouver, British Columbia Canada, August 2002,

[7] M. Aktaruzzaman, Literature *Review and Survey: Resource Discovery in Computational Grids*, School of Computer Science, University of Windsor, Windsor, Ontario, Canada.

[8] A. Andrieux, D. Berry, J. Garibaldi, S. Jarvis, J. MacLaren, D. Ouelhadj and D. Snelling, *Open Issues in Grid Scheduling*, Official Technical Report of the Open Issues in Grid Scheduling Workshop, Edinburgh, UK, October 2003.

[9] R. Bajaj and D. P. Agrawal, *Improving Scheduling of Tasks in A Heterogeneous Environment*, in IEEE Transactions on Parallel and Distributed Systems, Vol.15, no. 2, pp.107 – 118, February 2004.

[10] M. Baker, R. Buyya and D. Laforenza, *Grids and Grid Technologies for Wide-area Distributed Computing*, in J. of Software-Practice & Experience, Vol. 32, No.15, pp: 1437-1466, December 2002.

[11] F. Berman, R. Wolski, S. Figueria, J. Schopf and G. Shao, *Application-Level Scheduling on Distributed Heterogeneous Networks*, in Proc. of the 1996 ACM/IEEE Conference on Supercomputing, Article No: 39, Pittsburgh, Pennsylvania USA, November 1996.

[12] F. Berman, *High-Performance Schedulers*, chapter in The Grid: Blueprint for a Future Computing Infrastructure, edited by I. Foster and C. Kesselman, Morgan Kaufmann Publishers, 1998.

[13] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su and D. Zagorodnov, *Adaptive Computing on the Grid Using AppLeS*, in IEEE Trans. on Parallel and Distributed Systems (TPDS), Vol.14, No.4, pp.369--382, 2003.

[14] F. Berman, *High-Performance Schedulers*, chapter in The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, 1998.

[15] J. Bester, I. Foster, C. Kesselman, J. Tedesco and S. Tuecke, *GASS: A Data Movement and Access Service for Wide Area Computing Systems*, in Proc. of the 6th Workshop on I/O in Parallel and Distributed Systems, pp.: 78-88, Atlanta, Georgia USA, May 1999.

[16] J Blythe, S Jain, E Deelman, Y Gil, K Vahi and A Mandal,K Kennedy, *Task Scheduling Strategies for Workflow-based Applications in Grids*, in Proc. of International Symposium on Cluster Computing and Grid (CCGrid'05), pp.759-767, Cardiff, UK, May 2005.

[17] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen and R. Freund, *A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems*, in J. of Parallel and Distributed Computing, vol.61, No. 6, pp. 810-837, 2001.

[18] R. Buyya, J. Giddy, and D. Abramson, *An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications*, in Proc. of the 2nd International Workshop on Active Middleware Services (AMS 2000), pp. 221-230, , Pittsburgh, USA, August 2000.

[19] R. Buyya and D. Abramson and J. Giddy and H. Stockinger, *Economic Models for Resource Management and Scheduling in Grid Computing*, in J. of Concurrency and Computation: Practice and Experience, Volume 14, Issue.13-15, pp. 1507-1542, Wiley Press, December 2002.

[20] R. Buyya, D. Abramson, and S. Venugopal, The Grid Economy, in Proc. of the IEEE, Vol. 93, No. 3, pp. 698-714, IEEE Press, New York, USA, March 2005.

[21] J. Cao, S. A. Jarvis, S. Saini, G. R. Nudd, *GridFlow: Workflow Management for Grid Computing,* in Proc. of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid'03), pp.198-205, Tokyo, Japan, May 2003.

[22] H. Casanova, M. Kim, J. S. Plank and J. J. Dongarra, *Adaptive Scheduling for Task Farming with Grid Middleware*, in the International J. of High Performance Computing Applications, Vol. 13, No.3, pp.231-240, 1999.

[23] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments*, in Proc. of the 9th hetero-geneous Computing Workshop (HCW'00), pp. 349-363, Cancun, Mexico, May 2000.

[24] T. Casavant, and J. Kuhl, *A Taxonomy of Scheduling in General-purpose Distributed Computing Systems*, in IEEE Trans. on Software Engineering Vol. 14, No.2, pp. 141--154, February 1988.

[25] C. Cavanaugh, L. Welch, B. Shirazi, E. Huh and S. Anwar, *Quality of Service Negotiation for Distributed, Dynamic Real-Time Systems*, in Proc. of IPDPS2000, Cancun, Mexico, May 2000.

[26] S. J. Chapin, D. Katramatos, J. Karpovich and Andrew S. Grimshaw, *The Legion Resource Management System*, in Proc. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99), in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS '99), Lecture Notes in Computer Science, Vol. 1659, pp.162-178**,** San Juan, Puerto Rico, April 1999.

[27] C. Chekuri, and M. Bender, *An Efficient Approximation Algorithm for Minimizing Makespan on Uniformly Related Machines*, in J. of Algorithms vol. 41, no.2, pp. 212-224, November 2001.

[28] H. Chen and M. Maheswaran, *Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems*, in Proc. of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), pp. 88--97, Fort Lauderdale, Florida USA, April 2002.

[29] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt and M. Ripeanu, *Giggle: A Framework for Constructing Scalable Replica Location Services*, in Proc. of the ACM/IEEE Conference on Supercomputing, pp.1-17, Baltimore, Maryland USA, November 2002.

[30] F. Chudak and D. Shmoys, *Approximation Algorithms for Precedence Constrained Scheduling Problems on Parallel Machines that Run at Different Speeds*, in J. of Algorithms, vol.30, pp.323--343, 1999.

[31] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan and J. Dongarra, *New Grid Scheduling and Rescheduling Methods in the GrADS Project*, in Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), pp.199--206, Santa Fe, New Mexico USA, April 2004.

[32] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, *A Resource Management Architecture for Metacomputing Systems*, In D.G. Feitelson and L. Rudolph, editors, in Proc of the 4th Workshop on Job Scheduling

Strategies for Parallel Processing, LNCS Vol. 1459 pp. 62–82, Orlando, Florida USA, March 1998.

[33] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, *Grid Information Services for Distributed Resource Sharing*, in Proc. the 10th IEEE International Symposium on High- Performance Distributed Computing (HPDC-10), pp. 181-194, San Francisco, California, USA, August 2001.

[34] H. Dail, H. Casanova, and F. Berman, *A Decoupled Scheduling Approach for the GrADS Environment*, in Proc. 2002 ACM/IEEE conference on Supercomputing, pp.1-14, Baltimore, Maryland USA, November 2002.

[35] S. Darbha and D.P. Agrawal, *Optimal Scheduling Algorithm for Distributed Memory Machines*, in IEEE Trans. on Parallel and Distributed Systems, vol. 9, no. 1, pp. 87-95, January 1998.

[36] E. Deelman, J. Blythe, Y. Gil, C.l Kesselman, G. Mehta, K. Vahi, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda*, Mapping Abstract Complex Workflows onto Grid Environments*, in Journal of Grid Computing, Vol. 1, No. 1, pp 9--23, 2003.

[37] E. Deelman, J. Blythe, Y.Gil, C. Kesselman, G. Mehta, S. Patil, M.H. Su, K. Vahi and M. Livny, *Pegasus: Mapping Scientific Workflows onto the Grid*, in the Proc. of Grid Computing: Second European AcrossGrids Conference (AxGrids 2004), pp.11- 26, Nicosia, Cyprus, January 2004.

[38] E. Deelman, G. Singh, M. H.i Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta and K. Vahi, *Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems*, submitted to the Scientific Programming Journal, January 2005.

[39] F. Desprez and A. Vernois, *Simultaneous Scheduling of Replication and Computation for Bioinformatic Applications on the Grid*, in Proc. of workshop on Challenges of Large Applications in Distributed Environments (CLADE 2005), pp.66-74, North Carolina, USA, July 2005.

[40] C. Diot and A. Seneviratne, *Quality of Service in Heterogeneous Distributed Systems*, in Proc of the 30th International Conference on System Sciences (HICSS) Volume 5: Advanced Technology Track, pp. 238--253, Maui, Hawaii, USA, January 2003.

[41] A. Doboli and P. Eles, *Scheduling Under Data and Control Dependencies for Heterogeneous Architectures*, in Proc. of the International Conference on Computer Design (ICCD'98), pp.602-608, Austin, Texas USA, October 1998.

[42] H. El-Rewini, T. Lewis, and H..Ali, *Task Scheduling in Parallel and Distributed Systems*, ISBN: 0130992356, PTR Prentice Hall, 1994.

[43] C. Ernemann, V. Hamscher and R. Yahyapour, *Economic Scheduling in Grid Computing*, in Proc. of 8th Workshop on Job Scheduling Strategies for Parallel Processing, in conjunction with HPDC/GGF 5, Edinburgh, Scotland, UK, pp.128-152, July 2002.

[44] I. Foster, C. Kesselman and S. Tuecke, *The Anatomy of the Grid*: *Enabling Scalable Virtual Organizations*, in the International J. Supercomputer Applications, 15(3), pp. 200-220, fall 2001.

[45] I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[46] I. Foster, A. Roy and V. Sander, *A Quality of Service Architecture That Combines Resource Reservation and Application Adaptation*, in Proc. 8th Int. Workshop on Quality of Service, pp.181-188, Pittsburgh, PA, USA, June 2000.

[47] I. Foster, *What is the Grid? A Three Point Checklist*, in GRIDToday, July 20, 2002.

[48] I. Foster and A. Iamnitchi, *On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing*, in Proc. of 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, CA, USA, February 2003.

[49] I. Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems*, in the Proc. of IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS Vol. 3779, pp 2-13, Beijing, China, December 2005.

[50] N. Fujimoto and K. Hagihara, *Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid*, in Proc. of ICPP 2003, Kaohsiung, Taiwan, China, October 2003.

[51] N. Fujimoto and K. Hagihara, *Near-Optimal Dynamic Task Scheduling of Precedence Constrained Coarse-Grained Tasks onto a Computational Grid*, in Proc. of ISPDC 2003, Ljubljana, Slovenia, October 2003.

[52] J. Gehring and T. Preiss, *Scheduling a Metacomputer with Uncooperative Sub-schedulers*, in Proc. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes on Computer Science vol. 1659, pp. 179–201, San Juan, Puerto Rico, April 1999.

[53] A. Geras, *A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors*, in J. of Parallel and Distributed Computing, Vol. 16, No.4, pp.276-291, 1992.

[54] L. Gong, X. Sun and E. F. Watson, *Performance Modeling and Prediction of Nondedicated Network Computing*, in IEEE Transaction on Comptuers, Vol.51, No.9, pp. 1041-1055, September 2002.

[55] V. Hamscher, U. Schwiegelshohn, A. Streit, R. Yahyapour, *Evaluation of Job-Scheduling Strategies for Grid Computing*, in Proc. of GRID 2000 GRID 2000, First IEEE/ACM International Workshop, pp. 191-202, Bangalore, India, December 2000.

[56] L He, S.A. Jarvis, D.P. Spooner, D. Bacigalupo, G. Tan and G.R. Nudd, *Mapping DAG-based Applications to Multiclusters with Background Workload*, in Proc. of IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), pp. 855- 862, May 2005.

[57] X. He, X. Sun and G. Laszewski, *A QoS Guided Min-Min Heuristic for Grid Task Scheduling*, in J. of Computer Science and Technology, Special Issue on Grid Computing, Vol.18, No.4,pp.442--451, July 2003.

[58] E. Heymann, M. A. Senar, E. Luque and M. Livny, *Adaptive Scheduling for Master-Worker Applications on the Computational Grid*, in Proc. of the 1st

IEEE/ACM International Workshop on Grid Computing, Lecture Notes In Computer Science, Vol. 1971, pp.214-227, Bangalore, India, December 2000.

[59] D. S. Hochbaum and D. B. Shmoys, *A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach*, SIAM Journal of Computing, Vol.17, pp.539-551, 1988.

[60] E. Huedo, R. S. Montero and I. M. Llorente, *Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications*, in Proc. of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04) pp. 28-33, A Coruna Spain, February 2004.

[61] M. Islam, P. Balaji, P. Sadayappan and D. K. Panda, *QoPS: A QoS Based Scheme for Parallel Job Scheduling*, in the Proc. of JSSPP '03, Seattle, Washington USA, June 2003.

[62] M. Iverson and F. Ozguner, *Dynamic*, *Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment*, in Proc. of Seventh Heterogeneous Computing Workshop, pp. 70-78, Orlando, Florida USA, March 1998.

[63] B. Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J. Girard, R. Strachowski, S. Yu, *Enabling applications for Grid Computing with Globus*, Redbook, IBM, March 2003.

[64] H. A. James, *Scheduling in Metacomputing Systems*, Ph.D. Thesis, The Department Of Computer Science, University of Adelaide, Australia, 1999.

[65] H. Jin, X. Shi, W. Qiang and D. Zou, *An Adaptive Meta-scheduler for Data-Intensive Applications,* in Int. J. Grid and Utility Computing, Vol.1, No.1, pp.32-37, 2005.

[66] S. Kim and J. B. Weissman, *A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications*, in Proc. of the 2004 International Conference on Parallel Processing (ICPP'04), pp. 406-413, Montreal, Quebec Canada, August 2004.

[67] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. L. Wang, *Heterogeneous Computing: Challenges and Opportunities*, IEEE Computer, Vol. 26, No. 6, pp. 18-27, June 1993.

[68] K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak and J. Pukacki, *Improving Grid Level Throughput Using Job Migration And Rescheduling*, Scientific Programming vol.12, No.4, pp. 263-273, 2004.

[69] G. Lanfermann, G. Allen, T. Radke and E. Seidel, *Nomadic Migration: A New Tool for Dynamic Grid Computing*, in Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC'01), pp. 429-430, San Francisco, California USA, August 2001.

[70] J. K. Lenstra and A. H. G. Rinnooy Kan, *Complexity of Scheduling under Precedence Constraints*, Operations Research, vol. 26, pp. 22-35, 1978.

[71] J. Liou and M. A. Palis, *An Efficient Task Clustering Heuristic for Scheduling DAGs on Multiprocessors*, in Proc. of Workshop on Resource Management, Symposium of Parallel and Distributed Processing, 1996.

[72] J. Liou, and M. A. Palis, *A Comparison of General Approaches to Multiprocessor Scheduling*, in Proc. of the 11th International Symposium on Parallel Processing, p.152-156, April 1997.

[73] Y. Liu, *Survey on Grid Scheduling* (for Ph.D Qualifying Exam), Department of Computer Science, University of Iowa, http://www.cs.uiowa.edu/~yanliu/, April 2004.

[74] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke. *CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids*, in the Proc. of the ACM Java Grande 2000 Conference, pp.97-106, CA, USA, June 2000.

[75] Tianchi Ma and Rajkumar Buyya, *Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids*, in Proc. of the 17th International Symposium on Computer Architecture and High Performance Computing, Rio de Janeiro, Brazil, October 2005.

[76] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen and R. F. Freund, *Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems*, in J. of Parallel and Distributed Computing, Vol. 59, No. 2, pp.107--131, November 1999.

[77] G. Mateescu**,** *Quality of Service on the Grid via Metascheduling with Resource Co-Scheduling and Co-Reservation*, International Journal of High Performance Computing Applications, Vol. 17, No. 3, pp.209-218, 2003.

[78] S. McGough, L. Young, A. Afzal, S. Newhouse and J. Darlington, *Workflow Enactment in ICENI*, in Proc. of UK e-Science All Hands Meeting, pp. 894-900, Nottingham, UK, September 2004.

[79] N. Muthuvelu, J. Liu, N. L. Soe, S.r Venugopal, A. Sulistio and R. Buyya, *A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids*, Proceedings of the 3rd Australasian Workshop on Grid Computing and e-Research (AusGrid 2005), Newcastle, Australia, January 30 – February 4, 2005.

[80] K. C. Nainwal, J. Lakshmi, S. K. Nandy Ranjani Narayan, and K. Varadarajan, *A Framework for QoS Adaptive Grid Meta Scheduling*, in Proc. of the 16th International Workshop on Database and Expert Systems Applications, pp.292-296, Denmark, November 2005.

[81] Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesi_nska, Thilo Kielmann and Henri E. Bal, *Adaptive Load Balancing for Divide-and-Conquer Grid Applications*, http://www.cs.vu.nl/~kielmann/papers/satin-crs.pdf, submitted to the J. of Super-computing, 2004.

[82] S. Park and, J. Kim, *Chameleon: a Resource Scheduler in a Data Grid Environment*, in Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03), pp. 253-260, Tokyo Japan, 2003.

[83] A. Radulescu and A. J. C. van Gemund, *On the Complexity of List Scheduling Algorithms for Distributed Memory Systems*, in Proc. of 13th International Conference on Supercomputing, pp. 68-75, Portland, Oregon, USA, November 1999.

[84] S. Ranaweera and D. P. Agrawal, *A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems*, in Proc. of 14th International Parallel and Distributed Processing Symposium (IPDPS'00), pp. 445-450, Cancun, Mexico, May 2000.

[85] K. Ranganathan and I. Foster, *Identifying Dynamic Replication Strategies for a High Performance Data Grid*, in Proc. of the 2nd of International Workshop on Grid Computing, Lecture Notes In Computer Science; Vol. 2242, pp.75-86, Denver Colorado USA, November 2001.

[86] K. Ranganathan and I. Foster, *Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications*, in Proc. of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), pp. 352-361, Edinburgh, Scotland, July 2002.

[87] H. G. Rotithor, *Taxonomy of Dynamic Task Scheduling Schemes in Distributed Computing Systems*, in IEE Proc. on Computer and Digital Techniques, Vol.141, No.1, pp.1-10, January 1994.

[88] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan, *Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment*, in the Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes In Computer Science; Vol. 2862, Washington, U.S.A, June 2003.

[89] F.D. Sacerdoti, M.J. Katz, M.L Massie and D.E. Culler, *Wide area cluster monitoring with Ganglia*, in Proc. of IEEE International Conference on Cluster Computing, pp.289 – 298, Hong Kong, December 2003.

[90] R. Sakellariou and H. Zhao, *A Low-cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems*, in J. of Scientific Programming, Vol.12, No.4, pp. 253-262, 2004.

[91] R. Sakellariou and H. Zhao, *A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems*, in the Proc. of 18th International Parallel and Distributed Processing Symposium (IPDPS'04), pp.111-123, Santa Fe, New Mexico USA, April 2004.

[92] N. Sample, P. Keyani and G. Wiederhold, *Scheduling Under Uncertainty: Planning for the Ubiquitous Grid*, in Proc. of the 5th International Conference on Coordination Models and Languages, Lecture Notes In Computer Science; Vol. 2315, pp.300-316, York, UK, April 2002.

[93] J. Schopf and F. Berman, *Stochastic Scheduling*, in Proc. the 13th of Conference on SuperComputing, pp. 48-68, Portland, Oregon USA, November 1999.

[94] J. Schopf, *Ten Actions When SuperScheduling*, document of Scheduling Working Group, Global Grid Forum, http://www.ggf.org/documents/GFD.4.pdf, July 2001.

[95] H. Shan, L. Oliker, R. Biswas, and W. Smith, *Scheduling in Heterogeneous Grid Environments: The Effects of Data Migration*, in Proc. of ADCOM2004: International Conference on Advanced Computing and Communication, Ahmedabad Gujarat, India, December 2004.

[96] H. J. Siegel, H. G. Dietz, and J. K. Antonio, *Software Support for Heterogeneous Computing*, in ACM Computing Surveys, Vol.28, No.1, pp. 237 – 239, March 1996.

[97] D. P. Silva, W. Cirne and F. V. Brasileiro, *Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids*, in Proc of Euro-Par 2003, pp.169-180, Klagenfurt, Austria, August 2003.

[98] S. Song, Y. Kwok, and K. Hwang, *Security-Driven Heuristics and A Fast Genetic Algorithm for Trusted Grid Job Scheduling*, in Proc. of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), pp.65-74, Denver, Colorado USA, April 2005.

[99] D.P. Spooner, J. Cao, J.D. Turner, H. N. L. C. Keung, S.A. Jarvis and G.R. Nudd, *Localised Workload Management using Performance Prediction and QoS Contracts*, in the Proc. of the 18th Annual UK Performance Engineering Workshop (UKPEW' 2002), University of Glasgow, UK, July 2002.

[100]N. Spring and R. Wolski, *Application Level Scheduling of Gene Sequence Comparison on Metacomputers*, in the Proc. Of 1998 International Conference on Supercomputing (ICS'98), pp. 141-148, Melbourne, Australia, July 1998.

[101]A. Streit, *A Self-Tuning Job Scheduler Family with Dynamic Policy Switching*, in the Proc. of the 8th International Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes In Computer Science; Vol. 2537, pp. 1-23, Edinburgh, Scotland, July 2002.

[102]A. Su, F. Berman, R. Wolski, and M. Mills Strout, *Using AppLeS to Schedule Simple SARA on the Computational Grid*, in International J. of High Performance Computing Applications, vol. 13, no. 3, pp. 253-262, 1999.

[103]V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan, *Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests*, in Proc. of 11th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), pp.359- 366, Edinburgh, Scotland, July 2002.

[104]J. Subhlok, P. Lieu and B. Lowekamp, *Automatic Node Selection for High Performance Applications on Networks*, in Proc. of the 7th ACM SIGPLAN symposium on Principles and practice of parallel programming, pp163-172, Atlanta, Georgia USA, May1999.

[105]X. Sun and M. Wu, *Grid Harvest Service: A System for Long-term Application-level Task Scheduling*, in Proc. of 2003 International Parallel and Distributed Processing Symposium (IPDPS 2003), pp. 25--32, Nice, France, April 2003.

[106]A. Takefusa, S. Matsuoka, H. Casanova and F. Berman, *A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid*, in Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), pp. 406--415, San Francisco, California USA, August 2001.

[107]H. Topcuoglu, S. Hariri, M.Y. Wu, *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing*, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 3, pp. 260 - 274, 2002.

[108]S. Vadhiyar and J. Dongarra, *A Performance Oriented Migration Framework for the Grid*, in Proc. of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid'03), pp.130-139, Tokyo, Japan, May 2003.

[109]S. Venugopal and R. Buyya, *A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data Grids*, in Proc. of 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005), pp.60-72, Melbourne, Australia, Oct. 2-5, 2005.

[110]M. Wieczorek, R. Prodan and T. Fahringer, *Scheduling of Scientific Workflows in the ASKALON Grid Environment*, in ACM SIGMOD Record, Vol.34, No.3, pp.56-62, September 2005.

[111]R. Wolski, N. T. Spring and J. Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*, in the J. of Future Generation Computing Systems, Vol. 15, No. 5-6, pp. 757--768, January 1999.

[112]D. Wright, *Cheap Cycles from the Desktop to the Dedicated Cluster: Combining Opportunistic and Dedicated Scheduling with Condor*, in Proc. of Conference on Linux Clusters: the HPC Revolution, Champaign Urbana, IL USA, June 2001.

[113]M. Wu and X. Sun, *A General Self-Adaptive Task Scheduling System for Non-Dedicated Heterogeneous Computing*, in Proc. of IEEE International Conference on Cluster Computing (CLUSTER'03), pp.354-361, Hong Kong, December 2003.

[114]M. Wu and X. Sun, *Self-adaptive Task Allocation and Scheduling of Meta-tasks in Non-dedicated Heterogeneous Computing*, accepted in a special issue of the International J. of High Performance Computing and Networking (IJHPCN), 2004.

[115]M. Wu, W. Shu and H. Zhang, *Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems*, in Proc. of the 9th Heterogeneous Computing Workshop (HCW'00), pp. 375--385, Cancun, Mexico, May 2000.

[116]L. Yang, J. M. Schopf and I. Foster, *Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments*, in Proc. of the ACM/IEEE SuperComputing2003 Conference, pp.31-- 46, Phoenix, Arizona, USA, November 2003.

[117]T. Yang and A. Gerasoulis, *DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors*, in IEEE Trans. on Parallel and Distributed Systems, vol. 5, no. 9, pp.951--967, 1994.

[118]S.Y. You, H.Y. Kim, D. H. Hwang, S. C. Kim, *Task Scheduling Algorithm in GRID Considering Heterogeneous Environment*, in Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '04, pp. 240-245, Nevada, USA, June 2004.

[119]L. Young, S. McGough, S. Newhouse, and J. Darlington, *Scheduling Architecture and Algorithms within the ICENI Grid Middleware*, in Proc. of UK e-Science All Hands Meeting, pp. 5-12, Nottingham, UK, September 2003.

[120]J. Yu, R. Buyya, and C. K. Tham, *QoS-based Scheduling of Workflow Applications on Service Grids*, in Proc. of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science'05), Melbourne, Australia, December 2005,

[121]J. Yu and R. Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, accepted on Dec. 2005 by the J. of Grid Computing.

[122]H. Zhao and R. Sakellariou, *An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm*, in Proc. of Euro-Par 2003, Springer-Verlag, LNCS 2790, pp. 189-194, Klagenfurt, Austria, August 2003.

[123]Y. Zhu, *A Survey on Grid Scheduling Systems*, Department of Computer Science, Hong Kong University of science and Technology, 2003.

[124]Y. Zhu, L. Xiao, L. M. Ni and Z. Xu, *Incentive-based P2P Scheduling in Grid Computing*, in Proc. of the 3rd International Conference on Grid and Cooperative Computing (GCC2004), Wuhan, China, October 2004.

[125]http://www.globus.org

[126]http://www.openpbs.org

[127]http://www.cs.wisc.edu/condor