Resource Allocation for Query Processing in Grid Systems: A Survey

Deniz Cokuslu[1,2,3] denizcokuslu@iyte.edu.tr
Abdelkader Hameurlain[3] hameur@irit.fr
Kayhan Erciyes[4] kayhan.erciyes@izmir.edu.tr

[1]Izmir Institute of Technology, Department of Computer Engineering
Gulbahce, Urla, 35430 Izmir, Turkey
[2]International Computer Institute, Ege University
Bornova, 35100 Izmir, Turkey
[3]IRIT, Paul Sabatier University
118 Route de Narbonne, 31062 Toulouse, France
[4]Izmir University, Gursel Aksel Bulvari,
Uckuyular, 35350 Izmir, Turkey

Abstract

Grid systems are very useful platforms for distributed databases, especially in some situations in which the scale of data sources and user requests is very high. However, the main characteristics of grid systems such as dynamicity, large size and heterogeneity, bring new problems to the query processing domain such as resource discovery and resource allocation. In this paper, we provide a survey related to resource allocation methods for query processing in data grid systems. We provide a classification for existing studies considering their approaches to the resource allocation problem. We provide a synthesis of the studies and propose evaluations and comparisons for the different classes of studies.

Keywords: Resource allocation in grids, query processing, survey.

1. Introduction

Grid systems are one of today's most popular distributed computing environments, attracting the attention of both researchers and users. Their popularity comes mostly from their relaxed and large scale computing capabilities in which all users can run nearly all applications that they want. Grid systems differ from the classical distributed systems by their large scale, heterogeneous and dynamic nature [1]. The growth of grid systems, and their popularity, cause motivating middleware applications to be designed and implemented aiming at making grids more efficient and easier to use. They aim at providing user interfaces which hide whatever is happening in the background while running grid applications efficiently. However, the research density on such middleware applications dominates many other grid research since a complete grid middleware has not yet been designed which optimizes the use of grids in all different application domains [1]. For that reason, each different domain has specialized on its own optimization for grid systems independently from each other. Distributed database systems are one of those domains on which there exists a large amount of ongoing research [2, 3]. Grid systems are very useful platforms for distributed databases, especially in some situations in which the size of data sources and user requests are very large [4]. However, the main characteristics of grid systems, such as dynamicity, large size and heterogeneity, bring new problems such as resource discovery, resource selection, resource allocation, data management, autonomous computing, monitoring, replication and caching, security issues and many others [5]. In this study, we focus on Resource Allocation (RA) for query processing, which is a very important stage in efficient processing of queries in grid systems.

1.1. Background

Query processing is the core function for distributed data management. Queries may consist of relational algebra operators such as join, union, difference, projection and restriction. Each of these operators has a set of specialized tasks (scan, build, probe, etc.) which are waiting to be scheduled. Scheduling of these tasks on the suitable resources drastically influences the runtime of the queries. A resource in the grid may correspond to several different concepts. It may be a computational resource such as CPU, memory, storage unit or network; it may be a data resource, which provides metadata and its contents such as a database; or it may be a service, which is programmed to accomplish a specific task. On the other hand, a node corresponds to a computer in the grid which contains some of these resources with a set of characteristics. Regarding the definition of resources, RA for the query processing domain can be formally defined as a non-injective, non-surjective, multi-valued function from the set of tasks to the set of candidate resources, where tasks correspond to the operations (scan, build, probe, etc.), which compose operators in a query (join, union, difference, etc.).

After completion of the previous stage, resource discovery [6], a set of resources that are capable of executing the tasks is assumed known. At this point, in order to run these tasks effectively on the discovered resources, a resource allocation mechanism must be executed in order to determine: (i) how many of these resources will be used, (ii) which tasks will be executed by which resources and (iii) the action to be taken for dynamicity of resource properties during execution of the tasks. Many studies refer to different approaches for this purpose [3, 5, 7-21]. Existing studies are generally classified into two classes in the current literature: static resource allocation and dynamic resource allocation. In the static RA approach, allocation is performed once, and allocated nodes sustain execution until the tasks are completed. On the other hand, in the dynamic RA approach, according to the monitored status of the resources, allocation is dynamically modified during the execution of the tasks.

1.2. Objectives

The main objective of this paper is to provide a literature survey about recent resource allocation algorithms for query processing in grid systems by providing a classification, based on the most common attributes of the examined studies. In this way, we have classified the existing studies as *resource-centered resource allocation strategies* and *task-centered resource allocation strategies* according to whether their objective is to optimize resource utilization or task execution, respectively. For this classification, we have defined our scenario for the execution of the queries as follows. A grid user submits a database query to any node in the grid. The query is pre-processed in order to be optimized by the use of relational algebra before its execution. In this step, the query is decomposed into its smallest executable parts, which are called tasks in this paper, and a task dependency graph is generated in which vertices represent tasks and edges represent data flow between tasks. After finalizing the task dependency graph, a resource discovery phase is accomplished for each task in order to find all the candidate resources which have the capability to execute the corresponding task. After the resource discovery phase, resource allocation is realized in order to schedule tasks on the resources. Finding the optimum allocation scheme for the tasks has been proved to be NP-complete [22, 23], therefore in the resource allocation step, which is the focus of this paper, the aim is to find a near-optimum solution which will shorten the execution time of the query. We provide a synthesis of the recent resource allocation studies according to our classification. We also provide evaluations and comparisons between these classes considering both quantitative and qualitative criteria. For the quantitative evaluations and comparisons, we provide simulation results using the GridSim [24] simulator. For the qualitative evaluations and comparisons we examine the two classes according to the following criteria:

- Consideration of different task requirements: In a query, there may be different types of tasks, which have different requirements. For instance, one task may require a large amount of available memory space, whereas CPU characteristics may be much more important for another task. Therefore consideration of these different requirements for each task should be considered separately.
- Consideration of communication requirements between tasks: In queries, some tasks communicate with each other. In other words, the data produced by one task might be used by another task. Therefore data is transferred from one task to another in some situations. The resource allocation scheme must consider this communication, and hence precedence requirements between tasks, by taking into account the network characteristics between resources.
- Consideration of load balance: Load balancing is another important issue in allocating resources, because an allocation scheme which does not consider load balance on the resources might cause skew problems.
- Consideration of properties of resources: Since the grid environment is heterogeneous in its nature, different properties of resources, such as processor speed, available memory, internal bus speeds, network connections, etc., should be considered in order to allocate suitable resources to tasks. The precision of performance estimations of resources is tightly coupled with the level of detail of these properties.
- Scalability and reliability of the allocation method: Since the grid environment is considered to be large-scale and dynamic, the number of candidate resources and tasks might be very high and the nodes might be dynamic. Considering this fact, the proposed method should not contain bottlenecks and single point of failures.

In the rest of this paper, in Section 2, we examine recent survey studies in resource allocation for grid systems. In Section 3, we provide a detailed summary of recent resource allocation methods according to our classification. At the same time, we evaluate each study by considering qualitative evaluation criteria. In Section 4, we give qualitative comparisons between different classes. In Section 5, we provide quantitative evaluations and comparisons between the two proposed classes by simulation. Finally, in Section 6, we conclude our paper.

2. Existing Survey Studies and Motivations

Detailed survey studies can be found in the literature related to resource allocation and job scheduling in grid systems [25-29]. Even though we focus on resource allocation studies especially for query processing purposes, we believe that it is also essential to understand classical task scheduling and resource allocation in grid systems. For that reason, in this section, we examine existing survey studies, which are focused on both classical RA and RA for the query processing domain. There exist many survey studies related to classical RA in grid systems in the literature; we mention here those which may be applicable to the query processing domain [25-27]. On the other hand, we find very few survey studies which are focused on the query processing domain [28, 29].

In [25], Krauter et al. proposed a taxonomy and survey for resource management systems (RMS) in grid systems. The authors proposed different classifications for RMS according to machine organization within the grid, the resource model, dissemination protocols, namespace organization, data store organization, resource discovery, QoS (Quality of Service) support, scheduler organization, scheduling policy, state estimation, and the rescheduling approach. For the resource allocation phase, they proposed three different classes: centralized, hierarchical and decentralized resource allocation methods. Even though this survey is very detailed, evaluation criteria and comparison between different classes is not provided. In [26], Jian et al. presented a survey study for grid scheduling systems. They have examined the existing methods in three classes: computational economy, agent-oriented and service-oriented scheduling systems. Even though this

study is very interesting, detailed evaluation criteria for comparison between different methods are not introduced. Moreover, detailed analysis of the examined studies based on the standardized metrics is not examined. In [27], Jiang et al. proposed a survey study on job scheduling in grids. They examined and defined the grid specifications and evaluation criteria. However the study is mostly focused on security issues and fault tolerance. Classification between examined methods and a comparative study is not provided.

In [28], Costa et al. presented an experimental evaluation and comparison between different classes of grid RA algorithms which are focused on query scheduling. They examined classes according to the grid systems' characteristics. In their study, they examined schedulers in two classes: centralized and hierarchical grid query schedulers. Even though they provide comparisons based on experimental results, theoretical analyses and evaluation criteria are not presented. In another fruitful study [29], Epimakhov et al. proposed a survey study which examines resource scheduling methods for query optimization in data grid systems. They classify the existing methods as *extended classic* and *incentive based* approaches according to whether the method is extended from parallel and distributed systems or it is prepared from scratch for the grid environment. They provide analyses for each class and compare them by simulation.

Although these survey studies provide comprehensive and detailed background about resource allocation in grid systems, few survey studies are proposed which are focused on resource allocation specifically for query processing in data grid systems. It is crucial that, for analyses and survey studies in query processing domain, database queries and their characteristics should be particularly taken into consideration. For this reason, in this paper, we provide evaluation criteria considering particular requirements for database queries in grid systems. We provide a classification regarding the objectives of the existing approaches; and provide evaluations and comparisons between different classes by both qualitative and quantitative criteria. Hence, we believe that this study will be complementary to the existing survey studies in this domain [28, 29].

3. Recent Resource Allocation Methods

In this section, we provide a literature survey on recent resource allocation methods, which are designed for query processing in grid systems. We classified the studies according to their optimization goals. For each class of methods, we describe synthetically the main approach, followed by an evaluation and comparison with respect to the evaluation criteria, which are described in the Section 1.

3.1. Task-Centered Resource Allocation Methods

In task-centered resource allocation methods, the proposed studies examine the RA problem from the point of view of tasks. The primary objective is to minimize the task execution time independently of the state of the resources, such as load balance or resource utilization. The methods consider core characteristics of the resources like processor frequency, available memory, available disk space, network bandwidth etc. The main approach is to construct a ranking function for the resources by using these characteristics and allocate tasks to the most powerful resources aiming at minimizing the execution time. We examined the studies in two subclasses namely *static resource allocation* and *dynamic resource allocation*.

3.1.1 Synthesis and analysis

i. Methods based on static resource allocation:

In [10], Kant and Grosu proposed auction-based resource allocation protocols. In their model, each user has parameters for the total amount of work that is ready to be executed, the number of

tasks and the budget in grid dollars. The resources have parameters corresponding to processing rate, reservation price (minimum acceptable price), cost in grid dollars and resource profit indicating the profit gained by executing a task. They proposed three auction protocols using these parameters, namely *Preston-McAfee Double Auction (PMDA) protocol*, *Threshold Price Double Auction (TPDA) protocol* and *Continuous Double Auction (CDA) protocol*. In each protocol they determined different rules for the auction. At the end of the auction, the algorithm realizes resource allocation according to the matching of tasks and resources. The proposed study considers task requirements in terms of budget and total amount of work to be scheduled. Thus, different resource requirements between different tasks are not distinguished. The algorithm also does not consider communication requirements between tasks. It also does not take detailed resource properties into consideration such as available memory, workload and network connectivity. The suitability of resources for tasks and load balance is modeled by the use of prices of resources and bidding services. In this method, bidding services in resources and users are decentralized. Each user and resource has its own bidding service. However, the service which manages auctions is centralized. In a grid environment, the centralized service may become a bottleneck and single point of failure for the resource allocation service. This may negatively affect the scalability and reliability of the auctioning system.

Mandal et al. [11] proposed a workflow scheduling algorithm in grids. In their algorithm, they first rank the discovered nodes according to their expected performances for a specific application component. In the ranking phase, nodes are evaluated according to their properties which are provided by grid services: *MDS (Monitoring and Discovery Service)* [30] and *NWS (Network Weather Service)* [31]. Then a performance matrix is generated for every component in which performance estimations are stored. For each task, all possible candidate nodes are examined considering properties of resources and their weights. After constructing the matrix, the authors use three heuristics to allocate nodes: *min-min*, *max-min* and *suffrage* heuristics. The details of these heuristics can be found in [32]. After applying these heuristics, they choose the best performing one to allocate the nodes. Tasks in this study represent a workflow, and hence communication requirements between consecutive tasks are considered. On the other hand, the study considers different task requirements by the use of weighted parameters. The authors also consider detailed properties of resources, but they do not try to balance the load between the candidate nodes. In the proposed model, the performance matrix is built for each task by using the performance values of all nodes in the grid system. This may require handling of very large matrixes in large scale environments, which may drastically degrade the performance of the resource allocation process.

Gounaris et al. [7, 13] proposed a resource scheduling method for parallel query processing in computational grids. They examined the partitioned parallelism problem. In their algorithm, they assumed the existence of a parallel query plan. The algorithm starts its execution by an initial resource allocation according to the location of data. Then, for each operator, it checks if the parallelism degree of that operator can be increased. The algorithm allocates new resources until performance improvement drops below a threshold value. Each operator has selection criteria for choosing candidate resources. Many resource properties are considered by the use of these criteria which allow the algorithm to select appropriate resources for each task, but the load on the resources and load balancing is not mentioned in the cost model. In this study, different task requirements are considered by classifying the tasks, such as I/O bound, CPU bound, network bound etc. Although this approach does not allow the differentiation of the different task requirements in detail, it helps the algorithm to find suitable resources considering the common task requirements. This study is also focused on partitioned parallelism, thus, it does not consider communication requirements between different operators. Moreover, the algorithm is designed in a centralized manner, which may cause bottleneck and single point of failure problems in large-scale environments in which the number of queries and resources is high.

In [16], Bose et al. examined the problem of efficient resource allocation for a given set of parallel query sub-plans. They take advantage of the bushy tree representation of query plans. They

first divide the queries into two pieces by approximately balancing the load in each part. Then, they extracted sub-plans for the query which can be run in parallel and assign new nodes for those sub-plans. They recursively perform the above steps until a stopping criterion is satisfied and select new nodes by considering the estimated costs for execution of the sub-plans in candidate nodes. For the resource allocation phase, they generated a scheduling graph. For each sub-query, they have specific source and sink sub-plans as two outer nodes in the scheduling graph. In this graph, they present all possible resource scheduling combinations for the divided sub-plans as inner nodes. The graph contains the nodes' processing and communication capabilities. They perform resource allocation by selecting the least-cost path in this graph, which results in allocation of a node for the sub-plan. Since the algorithm considers all alternative allocations, it finds a near optimum scheduling for each query sub-plan. However, the algorithm does not distinguish between different requirements for each operator; instead, it aims at allocating the most powerful resource for query sub-plans in each cycle in a top-down fashion. This may result in inconvenient allocation for some operators with particular needs such as requirement of available memory space. On the other hand, it considers communication requirements between operators by examining bushy trees. The algorithm also considers different resource specifications such as network, processing speeds and load on resources but it does not distinguish the requirements by using weights. The time complexity of the algorithm is given as $O(\log K * n^3)$ where K is the total number of sub-plans in the query and n is the number of candidate nodes. Since in a grid environment n is assumed to be large, the algorithm may suffer from scalability issues in large-scale environments.

Liu and Karimi [18] developed a grid query optimizer for query processing in grids. They consider the resource discovery, resource allocation and query processing stages separately in their study. After discovering the available resources, their algorithm allocates nodes with the help of a ranking function. For the ranking function, they use the cost model which is proposed by Mackert and Lohman [33]. They use weighted parameters to evaluate resource properties such as CPU speed, available amount of memory, number of relations which are stored in the node, current workload of the node and estimated mean transmission latency. In their system, different services are responsible for providing resource information. The *Environment Information Service (EIS)* is responsible for providing information on computational resources. The *Database Information Service (DIS)* is responsible for determining the location of relations within nodes and the *Transmission Prediction Service (TPS)* is responsible for predicting transmission performances of candidate nodes. Liu and Karimi proposed a normalization method in predicting network transmission estimates, which is called *Transmission Latency Reputation*. This service prevents extremities from affecting negatively the estimations by using a reputation mechanism. These services provide required parameters related to nodes in order to calculate their ranks. For each different operator in the query, ranks for candidate nodes are calculated by using weighted ranking functions. The weights are set according to query requirements. After determining the rank of nodes for a given query operator, the node with the highest rank is allocated to the corresponding operator. This operation is repeated until all operators are mapped to a node. The proposed study considers different task requirements by the usage of adjustable weights for the ranking parameters. It also considers communication requirements between operators while allocating nodes. The proposed ranking functions include many resource properties including current workload on the resources. The authors showed that in a large-scale environment, the proposed algorithm outperforms an exhaustive search method because of its improved optimization complexity. Since the resource allocation process is performed by the querying node, the workload on this phase is distributed over the queried nodes, which avoids bottleneck problems. However, the static nature of the proposed allocation algorithm makes it prone to problems caused by the dynamicity of the grid environment.

ii. Methods based on dynamic resource allocation:

Silva et al. [14] presented an adaptive parallel query processing algorithm in which a resource allocation module is included. In their algorithm, after generating a parallel query plan and after

discovering nodes, they sort the candidate nodes according to their throughput. The calculation of the throughput is not detailed in the study. After sorting the candidate resources, they allocated all tasks to the first node in the sorted list. Then they move one task to the next node in the list. They analyze the performance of the query and continue doing so until no enhancement is achieved. After this initial allocation, the algorithm continuously validates and re-allocates nodes according to the changing environment. The proposed resource allocation algorithm uses a greedy approach which is based on the idea of allocating the fastest node for each task; therefore, instead of considering specific task requirements, it only considers the throughputs of the candidate resources. The throughput values of the resources are assumed to be provided by the grid middleware, thus the algorithm itself does not consider detailed properties of the resources. On the other hand, since the algorithm is designed to be dynamic, it helps load-balancing by modifying the allocation of overloaded nodes. It also considers communication requirements between tasks by constructing and examining task dependency graphs. The algorithm consists of different components in which the resource allocation component is centralized. Even though the query evaluator services are running independently within each allocated resources, the central scheduler service may become a single point of failure and bottleneck in a grid setting. Moreover, the communication overhead between the query execution manager and the query evaluators may cause excessive network traffic.

Venugopal et al. [15] proposed a grid service broker for scheduling e-science applications in data grids. They proposed an overall service which aims at discovering data and resources, allocating tasks to the resources, monitoring ongoing execution and accessing data during the execution of tasks. They focused on adaptive scheduling algorithms and brokering strategies. The resource allocation part of their study takes tasks and a set of candidate resources as input data. It then matches the task requirements with the services and allocates them to the remote nodes. For the tasks, which require access to the remote data, they use a network monitoring service to determine the status of the network. This information is used to optimize the scheduling strategy of the tasks according to the network status. In their system, they used *NWS (Network Weather Service)* [31] as the provider of the network monitoring service. During the initial allocation, they take the estimated task execution times into consideration. The authors do not consider detailed properties of resources; instead, they use a simplified statistics-based estimation model in estimating performance of the candidate resources. This approach does not consider load balance and detailed properties of resources. Network proximity is used as a primary metric in resource allocation; therefore the proposed study considers communication requirements for the tasks. Different task requirements are taken into account by the use of the requirement lists for tasks. In the proposed resource allocation method, the allocation is processed centrally by the grid service broker system, which may cause scalability and reliability problems in a large-scale dynamic environment such as the grid. Although this system is not designed especially for query processing, it aims at optimizing data-intensive tasks in data grids which makes it very suitable for the query processing domain.

3.1.2 Qualitative Evaluation

The task-centered resource allocation methods aim at minimizing query execution time by assigning tasks to the most powerful resources which are able to execute them. Since their perspective is from the tasks point of view, most of the studies in this class consider different task requirements, mostly by giving different weights for the resources' properties for each different task. In the examined studies, nearly all the static resource allocation based methods consider communication requirements of tasks by examining networking properties of resources; on the other hand, most of the dynamic resource allocation studies do not consider communication and precedence requirements of tasks as they are more focused on maintaining the optimum allocation in case of dynamicity of the environment. Since the proposed methods are designed from the point of view of task requirements, nearly all of the studies consider computational properties of nodes. In this class of algorithms, the most popular design paradigm is the greedy approach. In most of the

studies, the most powerful resources are allocated without performance considerations but by looking at the availability of sufficient resources. Cost functions in the mentioned studies consist of many resource properties, therefore, load balance is not considered as the primary constraint. Most of the studies in this class are designed in a centralized manner. Although heuristics are used in order to decrease complexity in the examined studies, in a grid setting, in which both number of resources and tasks can be very high, the centralized approach can negatively affect the scalability. It may also cause single point of failure problems in centralized allocation management, which decreases the reliability of the methods.

3.2. Resource-Centered Resource Allocation Methods

In resource-centered resource allocation methods, the studies look at the resource allocation problem from the point of view of resources. The primary aim in these studies is to maximize the resource utilization by considering both resource properties and load balance between them. The main idea here is that maximizing the resource utilization considering the sizes of tasks will maximize the throughput of the whole system. We examined this class in two subclasses as *static resource allocation* and *dynamic resource allocation* according to their allocation schemes. According to our search on this class of methods, we realized that most of the existing studies are concerned with dynamic resource allocation, although there are few static resource allocation studies as well. We examined here the studies which are most suitable for our domain: query processing in grid environments.

3.2.1 Synthesis and analysis

i. Methods based on static resource allocation:

Soe et al. [12] introduced a resource scheduling algorithm for parallel query processing in grids. They consider both inter-query and intra-query parallelism in scheduling resources. They consider the resource utilization ratios as the cost metric. Processor allocation is achieved level by level via iterative refinements. In their design, each query operator is firstly mapped to a processor considering the workloads of the resources; then the algorithm adjusts the load by moving assignments to processors from less costly operators to more expensive ones. For inter-query parallelism, they first analyze the query plans for an execution sequence. After extracting a sequence of the query plan, they map operators to the processors by considering estimated completion times. The study does not take communication requirements between operators into consideration when allocating resources. The resources are allocated according to their workloads taking into account the precedence order of the operators. This approach yields load balance between resources, however the computational and communicational properties of the resources are not considered. Therefore different task requirements are not distinguished. The examined algorithm is designed in a centralized manner, which can cause scalability problems in grid systems.

ii. Methods based on dynamic resource allocation:

In [5], Gounaris et al. proposed an algorithm for query processing in grids. The proposed algorithm is service oriented, thus it is considered to be autonomous. In this study, the initial resource allocation is achieved by the use of the algorithm which is proposed in [7]. Then, in each allocated node, a *grid query evaluation service (GQES)* is created. The *GQES* monitors the execution of the query sub-plan and creates notifications related to the current status of the local execution. A service named *Monitoring Event Detector* is created in each site, which evaluates the notifications. The *Diagnoser* component diagnoses the inappropriate workload and proposes an enhanced workload distribution vector to the *Responder* service. The *responder* service decides whether to move the execution to another candidate or to leave it at the current resource by

considering the status of the executions. The main objective in this algorithm is to make all participating resources finish at the same time. In case of a decrease in the performance of an execution, which may cause load imbalance, the algorithm makes a decision and moves the execution of the query sub-plan to another suitable node starting from the latest checkpoint. Since this study uses the core resource allocation scheme proposed in [7], it shares many common properties with it, such as consideration of different task requirements. But the dynamic nature of this study allows considering load balance between resources during the query execution. Similar to [7], this study does not consider communication requirements between different query sub-plans. In this algorithm, load balance is not only computed considering the workload in resources, but capabilities of resources are also considered. Therefore, a proportional load measure is proposed which allows the system to decrease makespan differences between query sub-plans. The algorithm is designed to be service oriented and decentralized, which makes it resilient to the dynamicity of the environment. It also allows the system to scale well with the number of nodes and queries.

Paton et al. [21] proposed a dynamic resource allocation algorithm for query processing in distributed large-scale dynamic environments. Their study aims at optimizing query execution by ensuring load balance between resources. During the execution of the query, they monitor the progress on the distributed portions of the process and according to the load differences, they redistribute the tasks to the resources. In their algorithm, they used a technique named *replicate operator state on demand,* which replicates relation fragments in case of load imbalances. More precisely, the algorithm replicates the hash table during the probe phase of join operators. In their paper they compared their algorithm with different approaches [5, 34-36]. Their study assumes that the underlying system is uniform; therefore suitability to grid systems is not clear. Since uniformity is assumed, the study does not consider heterogeneity between resources' computational or network capabilities. It also does not consider communication requirements between tasks. Since load balance is the primary concern in resource allocation, different task requirements are not considered in this study.

Kotowski et al. [3] introduced a parallel OLAP query processor in grid systems which exploits both intra and inter query processing. The proposed study is based on their previous database cluster middleware *ParGRES* [37], which exploits query parallelism and load balancing. By using the services provided by *ParGRES*, the authors developed the *GParGRES* system, which is a database grid middleware. *GParGRES* is designed as a wrapper that enables the use of *ParGRES* in a grid. They designed their system as a two-level database middleware in which *ParGRES* is responsible from node-level load balancing whereas *GParGRES* is responsible for grid-level load balancing. Since the design is focused only on load balancing, the detailed properties of resources and different task requirements are not considered in this study. Moreover, communication requirements between query sub-plans are not taken into consideration. On the other hand, the monitoring and allocation service in *GParGRES* is hierarchically decentralized in the grid, where node-level management is performed by *ParGRES* in resource sites and grid-level management is performed by *GParGRES* in site managers. This property increases the scalability of the system and decreases the risk of single points of failure.

In [19], Gounaris et al. proposed an adaptive query processing method in grid environments. The authors mention both data and state repartitioning in the distribution of data sources. The study is focused on dynamic resource allocation. The proposed approach is partially decentralized and its components communicate to each other by using a publish-subscribe model, which makes the algorithm suitable for autonomous environments such as grids. The study proposes adaptive query processing strategies for load balancing and bottleneck prevention. The load balance between allocated nodes is achieved by monitoring ongoing query executions and reacting to remarkable changes. The monitoring module is distributed to all allocated resource nodes. In contrast, decision-making modules are centralized for each query. The algorithm examines both communication and computational measures during monitoring. The proposed study also addresses the bottleneck problems by detecting overloaded resources and reacting in order to distribute load. It considers

bottlenecks on parallelizable operations. If the monitoring module notifies a slowdown of such an operator caused by a bottleneck problem, the responder checks the possible benefit of increasing the parallelization degree by using heuristics. If it is profitable, it increases the parallelism degree of the most costly operator working on the bottleneck resource, and distributes parallelized partitions to other resources. The study considers different task requirements by classifying tasks, but it does not consider communication requirements between them. The initial resource allocation, which is achieved by [13] based on [33], is performed centrally. However the rest of the dynamic allocation services are described as decentralized which increases the scalability and reliability of the system.

Ruiz et al. [20] proposed a query allocation method for distributed information systems. They defined the underlying environment as heterogeneous and dynamic. The proposed algorithm is based on a satisfaction model. For this purpose, they first defined a satisfaction model and then using this model they proposed a *satisfaction-based query load balancing (SQLB)*. *SQLB* is a flexible framework with self-adapting algorithms to allocate queries while considering both query load balancing and participants' goals. In this study, the existing information system is considered as a mediator system in which producers (resources) and consumers (tasks) post their capabilities and intentions respectively. The main role of the mediator is to assign tasks to the providers considering the resources' intentions. The study does not mention the cost metrics related to resources and tasks; instead, the authors propose generic metrics and give more attention to the balance between query requirements and resource intentions. The dynamicity of resources in terms of node failures is not covered. In this model, the allocation process is handled by a central mediator. It is claimed that the algorithm has a time complexity of $O(N \ Log \ N)$ for each query where N is the number of resources. The message complexity is expressed as $O(3(N+1))$ for each query. On the other hand, resource providers communicate to the mediator periodically in order to receive queued tasks, which might increase the network traffic where the number of resources is high. These factors might cause the central mediator to become a bottleneck and a single point of failure in crowded grid environments.

3.2.2 Qualitative Evaluation

The resource-centered resource allocation methods aim at balancing load between candidate resources. Nearly all mentioned studies focused on load balancing instead of task requirements. Therefore, the different requirements of tasks and communication requirements between them are not mentioned in many of these studies. On the other hand, some studies consider properties of resources in allocating tasks in order to calculate a proportional workload which can be considered as a more realistic metric. The key aim in this class of algorithms is that optimizing the load balance between resources will result in increased overall throughput. But the ignorance of communication requirements and different requirements of tasks may cause degraded performance. In such cases, even if the load is balanced between resources, the resource utilization may be dominated by processes that communicate to each other using slow communication links, which may be an inconvenient situation in terms of query optimization.

4. Qualitative Comparison Between Classes

The two different classes of studies, task-centered resource allocation and resource-centered resource allocation methods, aim at allocating resources in grid systems for query processing. Although these two classes share some common characteristics, they have many differences, which give advantages and disadvantages to each class of studies. A summary of global comparison can be seen in Table 1. In both classes, since the objective is resource allocation, the properties of resources are examined. On the other hand, in task-centered resource allocation methods, communication requirements between tasks, different task requirements and computational properties of resources have more priority than the load balance between resources. Therefore,

precise ranking functions are proposed in order to find near-optimal resource allocation schemes in this class of studies. On the other hand, this idea may easily cause unbalanced load distribution between resources, which may lead to skew problems between parallelized tasks. One can say that, if the different requirements of different tasks are uniformly distributed, the precise ranking functions may cause balanced resource allocation, but in reality, the distribution of task requirements should be considered separately in order to prove this assumption. On the other hand, in resource-centered resource allocation methods, the load balance between resources is assumed to lead to an increase in the overall throughput of the system, which is a true but an incomplete prediction. In our case, providing load balance is necessary but not sufficient to optimize the query execution since tasks within queries are dependent on each other. Therefore, besides load balancing, communication requirements between tasks should be considered as a function of environmental parameters such as network connections. Both approaches contain resource allocation methods based on static and dynamic allocations. Independently of being resource- or task-centered, the algorithms based on static allocation can be considered as fragile in a grid environment since dynamicity of resources may cause disruption of queries, whereas the algorithms based on dynamic allocation are more robust in terms of continuation or enhancement of the initial allocation's performance. Most of the examined studies in this survey propose centralized methods. Considering the characteristics of the grid systems, centralized methods may cause serious scalability and reliability problems caused by the dynamicity and large scale of the environment. Although decentralized methods deal with these problems, to the best of our knowledge, we cannot find a study that meets all the core characteristics of the grid systems in the distributed query processing domain.

Evaluation Criteria	***Task-Centered Methods***	***Resource-Centered Methods***
Consideration of different task requirements	Taken into account by the use of weights in ranking functions	Not taken into account in most of the studies
Consideration of communication requirements between tasks	Taken into account by the use of network requirement parameters for each task	Taken into account in terms of execution order, communication requirements are not considered
Consideration of load balance	Load balance is not taken into account	Load balance is considered to be the primary issue
Consideration of properties of resources	Taken into account by the use of detailed ranking functions	Not taken into account in most of the studies
Scalability and reliability	All studies are centralized, thus contain bottlenecks and single points of failure	Nearly all studies are centralized, thus contain bottlenecks and single points of failure. Those which are decentralized, do not suffer from scalability and reliability problems

Table 1. Qualitative Evaluation Summary

5. Quantitative Evaluation and Comparison

In this section, we provide quantitative evaluations and comparisons, between two classes of resource allocation methods, by simulation. Since there are many factors that affect a method's performance, we believe that it is not appropriate to implement and compare particular existing studies in this paper, as our implementations might not reflect their exact capabilities. Therefore, in our simulations, we implement an algorithm for each class, which uses basic parameters that reflect its classes' characteristics. With the selected parameters, we aim at obtaining meaningful results that allow comparison between two classes.

5.1 Simulation Setup

We have used the *GridSim* [24] grid simulator for our simulations. *GridSim* is a discrete event simulator, which is aimed at simulating task scheduling policies in grid environments. The resources in *GridSim* are modeled by considering their MIPS rating, number of processing units, loads and their connection speeds in baud rates.

In our simulation scenario, the tasks are treated as operations, which constitute query operators. In this manner, simple, medium and complex queries are assumed to correspond to 5, 10 and 20 tasks respectively. More than 20 tasks are considered as multiple independent queries that are waiting to be executed. During the simulations, all tasks are assumed to be independent. They are simulated according to their input data size, length in number of instructions, and output data size.

In the simulation setup, nodes are considered as uni-processor computers in the grid environment. The nodes, tasks and users are connected via simulated network links, which take into account baud rate, propagation delay and maximum transmission unit in bytes. The characteristics of nodes are set randomly using a uniform distribution within specific ranges. Tasks are also generated randomly in the same manner. Four different parameters are specified for the tests, namely, resource allocation type, number of nodes, number of tasks and CPU loads. For the resource allocation type, two different resource allocation methods are simulated: (i) Task-Centered Resource Allocation (TCRA) and (ii) Resource-Centered Resource Allocation (RCRA). In TCRA, the algorithm orders tasks according to their lengths, and nodes according to their reported MIPS ratings. Then nodes are allocated for the tasks in the same order as the lists in a round-robin fashion. In RCRA, the algorithm allocates the node with the least loaded CPU for each task. During the experiments, duration of resource allocation process and time required to execute all submitted tasks are measured. The measured values are in simulation time units and are used for comparison purposes.

5.2 Simulation Results, Qualitative Evaluations and Comparisons

In Figure 1, the time required for the resource allocation process is measured for scenario sizes varying between 10 and 500 nodes. Simulations are conducted for different scenarios consisting of 5 and 20 tasks. CPU loads of nodes are assigned randomly between 20% and 50%. The time required to complete the resource allocation process using different algorithms in different scenarios is plotted in Figure 1. Allocation times for 5 and 20 tasks in a small scale environment varying from 10 to 60 nodes is shown in Figure 1.a and Figure 1.b, respectively. In Figure 1.c and Figure 1.d, allocation times for 5 and 20 tasks in larger scale environments are shown, which consists of between 50 and 500 nodes.

The theoretical time complexity of the TCRA algorithm consists of (i) obtaining each node's characteristics for ranking, (ii) sorting nodes according to their ranks and (iii) sorting tasks according to their lengths. Thus it can be formalized as $O(m * c + n * \log n + m * \log m)$ where n is the number of tasks, m is the number of nodes and c is the communication overhead constant for obtaining the characteristics of a node. On the other hand, the time complexity of the RCRA algorithm consists of obtaining each nodes' dynamic load information for every single task. Thus it can be formalized as $O(n * m * c)$. In the complexity analysis of both algorithms, c is considered as the determining factor since it is considerably larger than other terms in the formulas because of the excessive communication costs. For that reason, the complexity of the algorithms depend on the term containing the constant c. The results of the simulations justify the complexity analyses of the algorithms. As can be seen in Figure 1, allocation times of both algorithms increase as the number of nodes increases. However, allocation time of the RCRA algorithm increases faster than TCRA. The speedup values between the two algorithms for this test can be seen in Figure 2.

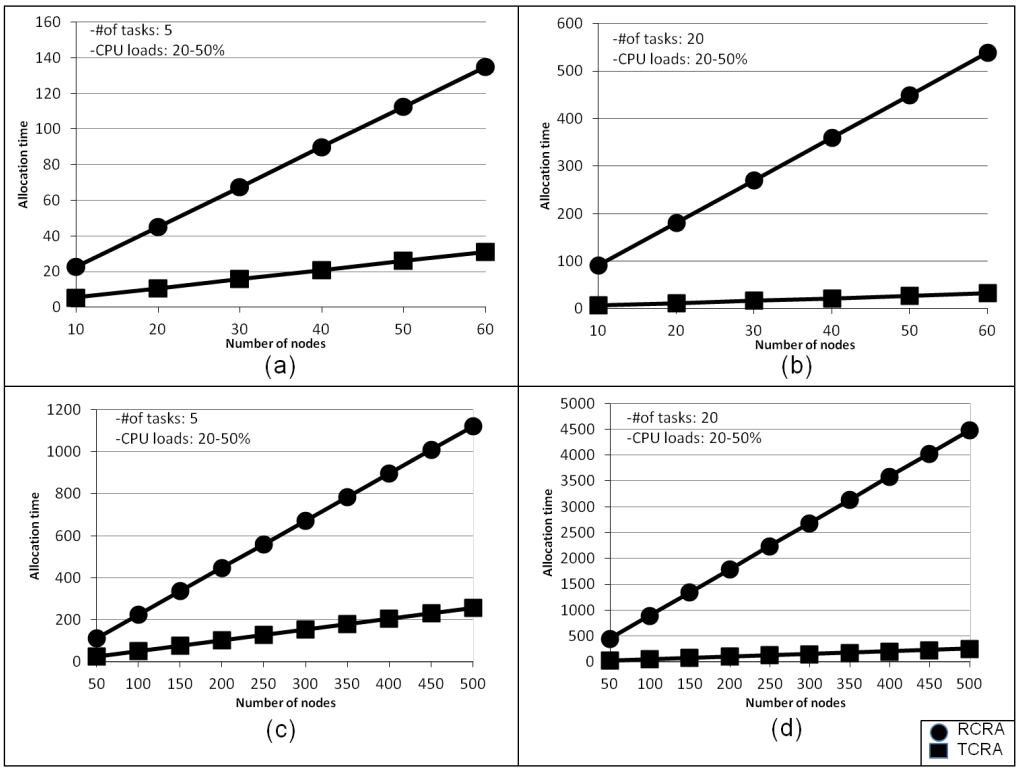

Figure 1. Allocation time for varying grid sizes.

It can be seen in Figure 2 that, as the number of nodes increases, the speedup value remains nearly constant since the speedup depends mostly on the number of tasks due to the communication costs. On the other hand, as the number of tasks increases, the speedup increases as a function of the number of tasks.

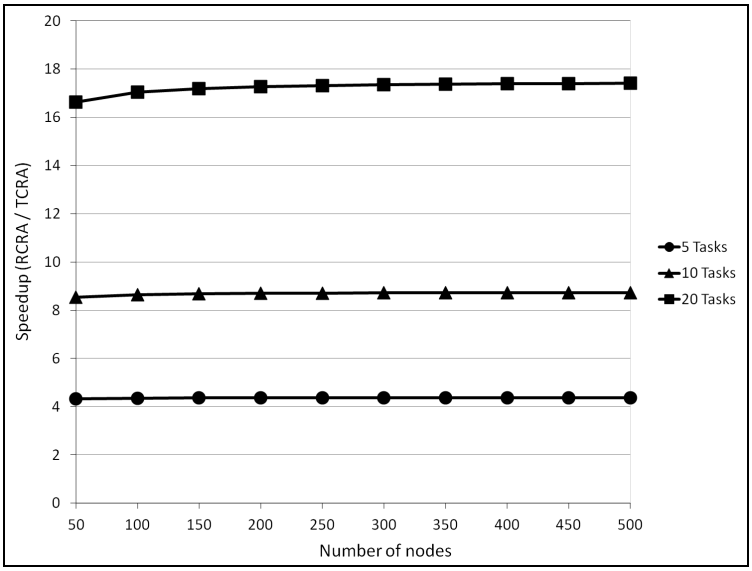

Figure 2. Speedup values for varying scale

In Figure 3, the time taken to execute all tasks during the simulation is measured for varying CPU loads in candidate nodes. The number of nodes is fixed at 200 and the number of tasks is fixed at 100. Task execution times are measured for varying node load densities in ranges 0-30%, 10-

40%, 20-50%,…, 70-100% and 80-100%. The results show that TCRA gives better results compared to RCRA in an environment in which CPU loads of nodes are smaller. However, for the cases where nodes are loaded more than nearly 70%, RCRA performs better in allocating better candidates for tasks.

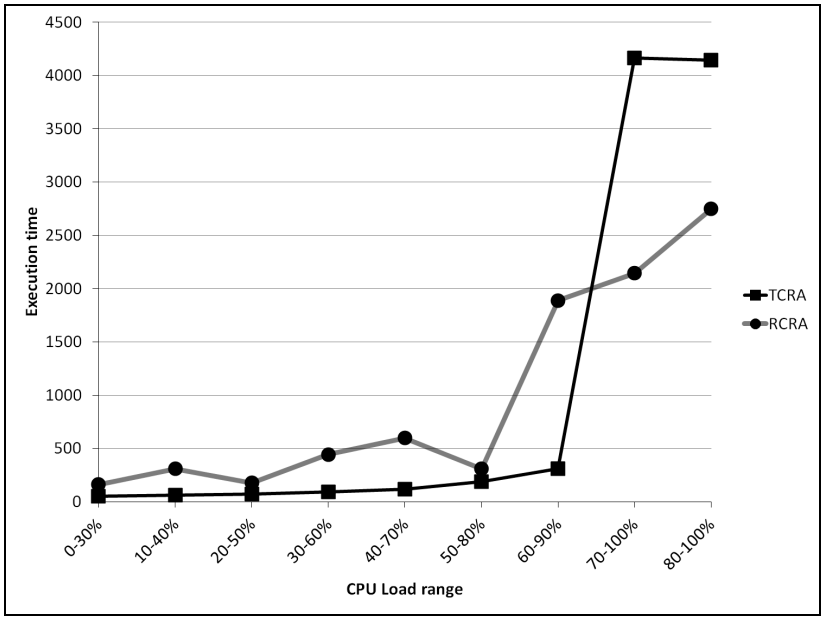

Figure 3. Execution time of tasks for varying CPU Loads

In Figure 4, we have examined how the execution time of tasks is affected by the changes in number of nodes. CPU loads are assigned to nodes uniformly distributed between 0% and 100%. Number of tasks were fixed at 175 in order to observe the behavior of different allocation methods in cases where: (C1) the number of nodes is smaller than the number of tasks, (C2) the number of nodes is greater than the number of tasks. As can be seen in Figure 4, RCRA performs better in case C1. This is because load balancing helps minimization of skew between task execution times. In case C2, TCRA performs better since the probability of finding powerful and less loaded nodes is increased. However, in case C1, the performance difference is more remarkable since in resource scarce environments, load balance gains more importance [38]. On the other hand, in case C2, the difference between the two approaches nearly converges to a constant after 250 nodes since sufficient resources can be found in any of such cases in which the number of resources is much higher than number of tasks.

Regarding the simulation results, RCRA can be considered to be favorable in cases in which the number of nodes is smaller than the number of tasks, and in cases in which the nodes are excessively busy. However, it must be noted that the complexity of RCRA is relatively high, and it might be a limiting factor in large-scale environments. On the other hand, TCRA performs comparatively better than RCRA in scenarios in which the number of nodes is higher than the number of tasks. It can be also be favorable in large-scale environments in which nodes are not excessively loaded.

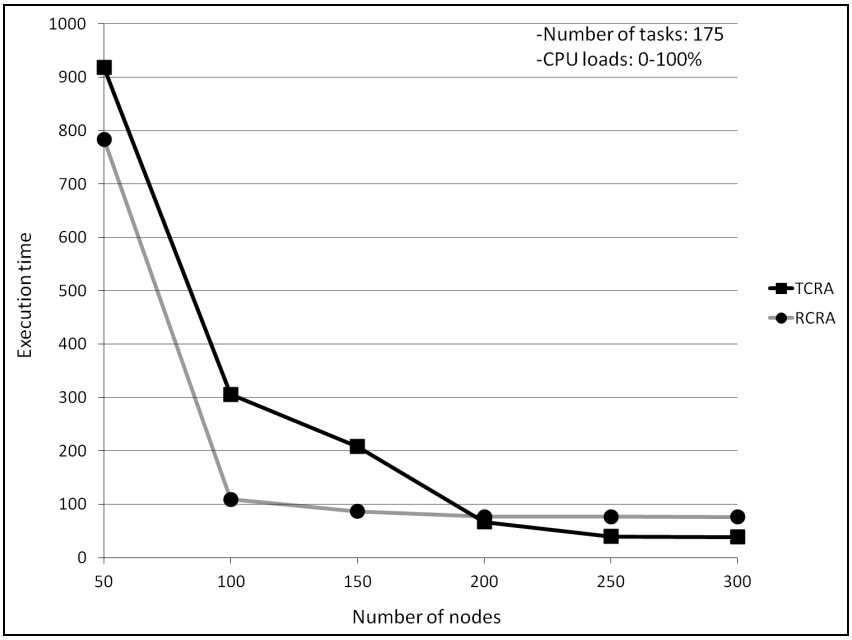

Figure 4. Execution time of tasks for varying number of nodes

6. Conclusion

In this paper, we classified, analyzed, evaluated and compared different kinds of resource allocation methods in grids. We provided a classification by considering the methods' viewpoints to the resource allocation problem. We have evaluated the different classes of methods considering important criteria for the query processing domain in grids. Finally we compared different classes and presented some remarks regarding their evaluations. Even though we have examined many fruitful methods in this paper, we cannot find studies that provide a complete solution regarding both the grid environments' characteristics and query processing requirements.

Both classes, which are examined in this paper, have advantages and disadvantages compared to each other; therefore it is not possible, at this stage, to conclude that one is superior to the other. Instead, they can be provided as alternative resource allocation methods in systems such as parametric query optimization [39, 40] in order to contribute to the query optimization by choosing the appropriate allocation method for the specific situation.

We believe that the proposed survey study may lead the researchers in this field to search for new methods to bring the two different classes' advantages together, especially in the distributed query processing domain.

References

1. Foster, I, Jennings, N R, Kesselman, C (2004). *Brain Meets Brawn: Why Grid and Agents Need Each Other*. AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, IEEE Computer Society.
2. Stonebraker, M, et al. (1996). Mariposa: a wide-area distributed database system. *The VLDB Journal*, **5** 048-063.
3. Kotowski, N, et al. (2008). Parallel query processing for OLAP in grids. *Concurr. Comput. : Pract. Exper.*, **20** 2039-2048.
4. Hameurlain, A, Morvan, F, Samad, M E (2008). *Large Scale Data management in Grid Systems: a Survey*. IEEE International Conference on Information and Communication Technologies: from Theory to Applications (ICTTA), Damas - Syrie, 07/04/2008-11/04/2008, IEEE.
5. Gounaris, A, et al. (2005). *Adapting to Changing Resource Performance in Grid Query Processing*. DMG,

Lecture Notes in Computer Science, Springer.
6. Hameurlain, A, Cokuslu, D, Erciyes, K (2010). Resource discovery in grid systems: a survey. *International Journal of Metadata, Semantics and Ontologies*, **5** 251-263.
7. Gounaris, A, et al. (2004). *Resource Scheduling For Parallel Query Processing On Computational Grids*. Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on,
8. Slimani, Y, Najjar, F, Mami, N (2004). *An Adaptive Cost Model for Distributed Query Optimization on the Grid*. OTM Workshops,
9. Buyya, R, Abramson, D, Venugopal, S (2005). The Grid Economy. *Proceedings of the IEEE*, **93** 698-714.
10. Kant, U, Grosu, D (2005). *Double Auction Protocols for Resource Allocation in Grids*. ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume I, IEEE Computer Society.
11. Mandal, A, et al. (2005). *Scheduling Strategies For Mapping Application Workflows Onto The Grid*. Proceedings, 14th IEEE International Symposium on High Performance Distributed Computing,
12. Soe, K M, et al. (2005). *Efficient Scheduling of Resources for Parallel Query Processing on Grid-based Architecture*. Information and Telecommunication Technologies, 2005. APSITT 2005 Proceedings. 6th Asia-Pacific Symposium on,
13. Gounaris, A, et al. (2006). A novel approach to resource scheduling for parallel query processing on computational grids. *Distrib. Parallel Databases*, **19** 87-106.
14. Silva, V F V D, et al. (2006). An adaptive parallel query processing middleware for the Grid: Research Articles. *Concurr. Comput. : Pract. Exper.*, **18** 621-634.
15. Venugopal, S, Buyya, R, Winton, L (2006). A Grid service broker for scheduling e-Science applications on global data Grids: Research Articles. *Concurr. Comput. : Pract. Exper.*, **18** 685-699.
16. Bose, S K, Krishnamoorthy, S, Ranade, N (2007). *Allocating Resources to Parallel Query Plans in Data Grids*. GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing, IEEE Computer Society.
17. Zhao, X, Xu, L, Wang, B (2007). *A Resource Allocation Model with Cost-Performance Ratio in Data Grid*. SNPD '07: Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, IEEE Computer Society.
18. Liu, S, Karimi, H A (2008). Grid query optimizer to improve query processing in grids. *Future Gener. Comput. Syst.*, **24** 342-353.
19. Gounaris, A, et al. (2009). Adaptive workload allocation in query processing in autonomous heterogeneous environments. *Distrib. Parallel Databases*, **25** 125-164.
20. Quian\'e-Ruiz, J-A, Lamarre, P, Valduriez, P (2009). A self-adaptable query allocation framework for distributed information systems. *The VLDB Journal*, **18** 649-674.
21. Paton, N W, et al. (2009). Autonomic query parallelization using non-dedicated computers: an evaluation of adaptivity options. *The VLDB Journal*, **18** 119-140.
22. Fernandez-Baca, D (1989). Allocating modules to processors in a distributed system. *Software Engineering, IEEE Transactions on*, **15** 1427-1436.
23. Chihping, W (1990). The complexity of processing tree queries in distributed databases. Conference Name, Conference Location,
24. Sulistio, A, et al. (2008). A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurr. Comput. : Pract. Exper.*, **20** 1591-1609.
25. Krauter, K, Buyya, R, Maheswaran, M (2002). A taxonomy and survey of grid resource management systems for distributed computing. *Softw. Pract. Exper.*, **32** 135-164.
26. Jian, Y, Liu, Y (2007). *The State of the Art in Grid Scheduling Systems*. Proceedings of the Third International Conference on Natural Computation - Volume 04, IEEE Computer Society.
27. Jiang, C, et al. (2007). *A survey of job scheduling in grids*. Proceedings of the joint 9th Asia-Pacific web and 8th international conference on web-age information management conference on Advances in data and web management, Huang Shan, China, Springer-Verlag.
28. Costa, R L d C, Furtado, P (2008). *Scheduling in Grid Databases*. Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops, IEEE Computer Society.
29. Epimakhov, I, et al. (2011). Resource Scheduling Methods for Query Optimization in Data Grid Systems
Advances in Databases and Information Systems. Eder, J., Bielikova, M., Tjoa, A., Springer Berlin / Heidelberg.
30. Foster, I, Kesselman, C (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of*

Supercomputer Applications, **11** 115-128.
31. Wolski, R, Spring, N T, Hayes, J (1999). The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Gener. Comput. Syst.*, **15** 757-768.
32. Braun, T D, et al. (2001). A Comparison Of Eleven Static Heuristics For Mapping A Class Of Independent Tasks Onto Heterogeneous Distributed Computing Systems. *J. Parallel Distrib. Comput.*, **61** 810-837.
33. Mackert, L F, Lohman, G M (1986). *R* optimizer validation and performance evaluation for local queries*. SIGMOD '86: Proceedings of the 1986 ACM SIGMOD international conference on Management of data, ACM.
34. Shah, M A, et al. (2003). Flux: an adaptive partitioning operator for continuous query systems. Conference Name, Conference Location,
35. Raman, V, Han, W, Narang, I (2005). *Parallel querying with non-dedicated computers*. Proceedings of the 31st international conference on Very large data bases, Trondheim, Norway, VLDB Endowment.
36. Paton, N W, et al. (2006). *Autonomic Query Parallelization using Non-dedicated Computers: An Evaluation of Adaptivity Options*. ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing, IEEE Computer Society.
37. Mattoso, M, et al. (2005). ParGRES: a middleware for executing OLAP queries in parallel. Conference Name, Conference Location, COPPE/UFRJ Technical Report ES-690.
38. Pourebrahimi, B, Bertels, K (2007). *Fair access to scarce resources in ad-hoc grids using an economic-based approach*. Proceedings of the 5th international workshop on Middleware for grid computing: held at the ACM/IFIP/USENIX 8th International Middleware Conference, Newport Beach, California, ACM.
39. Bizarro, P, Bruno, N, DeWitt, D J (2009). Progressive Parametric Query Optimization. *IEEE Trans. on Knowl. and Data Eng.*, **21** 582-594.
40. Ioannidis, Y E, et al. (1997). Parametric query optimization. *The VLDB Journal*, **6** 132-151.

View publication stats