Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Cotutelle Internationale avec Université Ege, Turquie

Présentée et soutenue par :

Deniz ÇOKUSLU

Le 02 Novembre 2012

Titre :

Découverte et allocation des ressources pour le traitement de requêtes dans les systèmes grilles

---

ED MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de recherche :

Institut de Recherche en Informatique de Toulouse

Directeur(s) de Thèse :

Abdelkader HAMEURLAIN, Professeur à l'Université Paul Sabatier Toulouse III, France
Kayhan ERCİYEŞ, Professeur à l'Université Izmir, Turquie

Rapporteurs :

Djamal BENSLIMANE, Professeur à l'Université Claude Bernard Lyon, France
Nihan Kesim ÇİÇEKLİ, Professeur à L'Université Technique du Moyen-Orient, Turquie

Autre(s) membre(s) du jury :

Farouk TOUMANI, Professeur à l'Université Clermont Ferrand II, France
Mehmet Emin DALKILIÇ, Professeur à l'Université Ege, Turquie
Franck MORVAN, Professeur à l'Université Paul Sabatier Toulouse III, France
Aysegul ALAYBEYOĞLU, Professeur Adjoint à l'Université Celal Bayar, Turquie

ACKNOWLEDGEMENTS

I owe my deepest gratitude to my supervisors Prof. Kayhan Erciyes and Prof. Abdelkader Hameurlain for their valuable guidance during this cooperation. Their priceless efforts make this joint thesis possible between Ege University and Paul Sabatier University. I would never converge to the end of this thesis without their precise foresight. I also wish to express my sincere gratitude to Prof Sıtkı Aytaç, Director of Department of Computer Engineering, İYTE, Prof M. Emin Dalkılıç, Director of International Computer Institute, Ege University and Prof Michel Daydé, Director of IRIT, Paul Sabatier University for their valuable cooperation during this thesis. I sincerely thank to Prof Nihan Kesim Çiçekli and Prof Djamal Benslimane for accepting to be the reviewers of this thesis and for their valuable remarks, which make this thesis more accurate. Likewise, I wish my sincere gratitude to the members of the jury of the thesis dissertation, Prof Kayhan Erciyes, Prof Abdelkader Hameurlain, Prof M. Emin Dalkılıç, Prof. Franck Morvan, Prof Djamal Benslimane, Prof Nihan Kesim Çiçekli, Prof Farouk Toumani and Assist. Prof. Ayşegül Alaybeyoğlu. I would like to thank to Assoc. Prof. Aylin Kantarcı and Prof Mehmet Emin Dalkılıç for accepting to be in my thesis advisory committee. I also would like to gratefully acknowledge The Scientific and Technological Research Council of Turkey TÜBİTAK and French Government for their financial support, which make this joint thesis possible. I would like to show my greatest appreciation to all my friends. I especially thank to Mr Selçuk Sözen, who believed in me by heart throughout my thesis. Last but not least I wish to avail myself of this opportunity, express a sense of gratitude and love to my beloved parents for their manual support, strength and help. Lastly, I take immense pleasure in thanking to my precious wife Evşen for her support and encouragements during preparation of this thesis.

Découverte et allocation des ressources pour le traitement de requêtes dans les systèmes grilles

Résumé

De nos jours, les systèmes Grille, grâce à leur importante capacité de calcul et de stockage ainsi que leur disponibilité, constituent l'un des plus intéressants environnements informatiques. Dans beaucoup de différents domaines, on constate l'utilisation fréquente des facilités que les environnements Grille procurent.

Le traitement des requêtes distribuées est l'un de ces domaines où il existe de grandes activités de recherche en cours, pour transférer l'environnement sous-jacent des systèmes distribués et parallèles à l'environnement Grille.

Dans le cadre de cette thèse, nous nous concentrons sur la découverte des ressources et des algorithmes d'allocation de ressources pour le traitement des requêtes dans les environnements Grille. Pour ce faire, nous proposons un algorithme de découverte des ressources pour le traitement des requêtes dans les systèmes Grille en introduisant le contrôle de topologie auto-stabilisant et l'algorithme de découverte des ressources dirigé par l'élection convergente. Ensuite, nous présentons un algorithme d'allocation des ressources, qui réalise l'allocation des ressources pour les requêtes d'opérateur de jointure simple par la génération d'un espace de recherche réduit pour les nœuds candidats et en tenant compte des proximités des candidats aux sources de données. Nous présentons également un autre algorithme d'allocation des ressources pour les requêtes d'opérateurs de jointure multiple. Enfin, on propose un algorithme d'allocation de ressources, qui apporte une tolérance aux pannes lors de l'exécution de la requête par l'utilisation de la réplication passive d'opérateurs à état.

La contribution générale de cette thèse est double. Premièrement, nous proposons un

nouvel algorithme de découverte de ressource en tenant compte des caractéristiques des environnements Grille. Nous nous adressons également aux problèmes d'extensibilité et de dynamicité en construisant une topologie efficace sur l'environnement Grille et en utilisant le concept d'auto-stabilisation, et par la suite nous adressons le problème de l'hétérogénéité en proposant l'algorithme de découverte de ressources dirigé par l'élection convergente. La deuxième contribution de cette thèse est la proposition d'un nouvel algorithme d'allocation des ressources en tenant compte des caractéristiques de l'environnement Grille.

Nous abordons les problèmes causés par la grande échelle caractéristique en réduisant l'espace de recherche pour les ressources candidats. De ce fait nous réduisons les coûts de communication au cours de l'exécution de la requête en allouant des nœuds au plus près des sources de données. Et enfin nous traitons la dynamicité des nœuds, du point de vue de leur existence dans le système, en proposant un algorithme d'affectation des ressources avec une tolérance aux pannes.

Mots-clés: Traitement des requêtes, découverte de ressources, contrôle de topologie auto-stabilisant, arbres couvrants, allocation des ressources, tolérance aux pannes.

Resource Discovery and Allocation for Query Processing in Grid Systems

Abstract

Grid systems are today's one of the most interesting computing environments because of their large computing and storage capabilities and their availability. Many different domains profit the facilities of grid environments. Distributed query processing is one of these domains in which there exists large amounts of ongoing research to port the underlying environment from distributed and parallel systems to the grid environment.

In this thesis, we focus on resource discovery and resource allocation algorithms for query processing in grid environments. For this, we propose resource discovery algorithm for query processing in grid systems by introducing self-stabilizing topology control and converge-cast based resource discovery algorithms. Then, we propose a resource allocation algorithm, which realizes allocation of resources for single join operator queries by generating a reduced search space for the candidate nodes and by considering proximities of candidates to the data sources. We also propose another resource allocation algorithm for queries with multiple join operators. Lastly, we propose a fault-tolerant resource allocation algorithm, which provides fault-tolerance during the execution of the query by the use of passive replication of stateful operators.

The general contribution of this thesis is twofold. First, we propose a new resource discovery algorithm by considering the characteristics of the grid environments. We address scalability and dynamicity problems by constructing an efficient topology over the grid environment using the self-stabilization concept; and we deal with the heterogeneity problem by proposing the converge-cast based resource discovery algorithm. The second main contribution of this thesis is the proposition of a new resource allocation algorithm considering the characteristics of the grid environment. We tackle the scalability problem

by reducing the search space for candidate resources. We decrease the communication costs during the query execution by allocating nodes closer to the data sources. And finally we deal with the dynamicity of nodes, in terms of their existence in the system, by proposing the fault-tolerant resource allocation algorithm.

Keywords: Query processing, resource discovery, self-stabilization, topology control, spanning tree, resource allocation, fault-tolerance.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Résumé

Dans ce chapitre, nous présentons nos motivations de cette thèse. Nous présentons les définitions des problèmes abordés. Nous décrivons brièvement un état de l'art et nous discutons nos contributions potentielles. Enfin, nous présentons l'idée principale derrière les algorithmes proposés ainsi que la structure de la thèse.

Abstract

In this chapter, we provide motivations for this thesis. We present the tackled problem definitions. We describe a state of the art very briefly and discuss the open issues and our potential contributions. Finally, we present the main idea behind the proposed algorithms and also the structure of the thesis.

1.1 Context, Motivations and Problems

Research activities are driven by new applications, technology trends, and innovative aspects. Since the 90s, the Internet becomes the most important driving force for scientific application development. As the technological development progresses, complex systems are developed in order to manage increasing large amounts of data, such as database management systems (DBMS). Those systems provide fast and reliable management methods to respond increasing demand for the data. The first database management systems are designed to work on centralized servers which both manage and store the data [DEWITT84]. However, with the increase in the scale of both data and users, the centralized DBMSs become inadequate. To overcome this problem, the researchers directed their attention to Distributed DBMS (DDBMS) design [OZSU11]. Distributing the DBMS over the network gives many flexibilities and scaling properties to the information technology. After distribution of the data, researchers directed their attention to increase the flexibility of those systems by integrating different types of data sources such as

DBMSs, XMLs, File Systems etc. This research makes out a new concept, mediation systems, which can be defined as systems which are focused on the integration of heterogeneous data sources in a large scale distributed environment [OZSU11].

The advances in the information technology bring forth together the increase in the need for shared resources in the large-scale domain. This fact leads researchers to propose inter-connected virtual organizations, called Grids. Grid system is a large scale distributed environment which provides high number of powerful resources to its users [FOSTER04b]. The main objective of Grids is to provide a powerful and robust platform that serves those resources without being affected by the dynamicity of the nodes. The resources offered by Grids have to be accessible to the users easily without any deep technical knowledge. On the other hand, the management services such as job scheduling, load balancing, resource discovery and allocation must be realized in the background without user interaction. The grid systems are characterized by their dynamic, large scale and heterogeneous nature. The grid environment is large scale in terms of number of users, number of requests and number of resources. The resources are heterogeneous since they are not forced to be identical in grid environment. Moreover, the network presents a low bandwidth and a strong latency in most cases. A resource in the Grid may correspond to several different concepts. It may be a computational resource such as CPU, memory, storage unit, network; it may be a data source that provides metadata and its content such as 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 those resources with a set of characteristics. The nodes in grid systems are dynamic in terms of dynamicity of their properties and dynamicity of their existence in the grid. At any time, there may be new nodes joining to the grid, or there may be some nodes that leave the system without any notice [FOSTER04b]. Under these characteristics, Grids aim to provide their users a large number of resources. Grid systems are widely used in today's research and industry communities. For instance, computational scientists use to realize their heavy computational studies such as complex simulations, artificial intelligence,

image processing etc. Experimental researchers use grids to process experimental data that can have enormous sizes such as Large Hadron Collider (LHC) experiment in CERN. Corporations and industry also use grid resources to process and manage their high volumes of corporate data [FOSTER04b].

Due to the fact that the usage area of grid systems increases every day, the need for the adaptation of some existing concepts such as mediation systems to the grid environments is inevitable. The main difficulties in the adaptation period are caused by the different characteristics of the underlying environment. The main assumptions that exist in the distributed and parallel mediation systems do not hold in grid environments. Those assumptions can be listed as: *i)* nodes are homogenous, *ii)* system is stable and *iii)* there is a limited number of resources. The invalidity of those assumptions and characteristics of grid systems generate new critical problems and challenges in the extension of distributed and parallel systems to grids from the mediation systems' point of view, especially in the distributed query-processing domain. Some of these problems, particularly in the domain of distributed query processing, can be listed as: *i)* resource discovery, *ii)* resource selection and task scheduling, *iii)* autonomous computing and monitoring services, *iv)* replication and caching, *v)* security issues and others. From those, resource discovery and resource allocation are two of the most attractive topics for today's research community. For that reason, in this thesis, we focus on resource discovery and resource allocation problems in the domain of distributed query processing in grid environments.

1.2 Resource Discovery (RD) for Query Processing in Grid Environments

Resource discovery in grids can be defined as searching and locating resource candidates that are suitable for a job in which processing environments' constraints are clearly specified. Resource discovery should be realized in a reasonable time by sticking the reliability restrictions considering heterogeneity, dynamicity and scale of the grid system. In this perspective, several methods have been proposed for RD problem in grid environments. Although there can be found many different classifications, existing studies

are generally classified into three main categories [HAMEURLAIN08, HAMEURLAIN10]: methods based on centralized/hierarchical systems [YU03, FOSTER04b, ANTONIOLETTI05, ELMROTH05, RAMOS06, KAUR07, MOLTO07], methods based on the peer to peer (P2P) systems [CAI03, CHEEMA05, MARZOLLA07, TRUNFIO07], and methods based on agent systems [CAO02, DING05, KAKARONTZAS06, YAN07, YU06]. Although these studies provide fruitful solutions to the RD problem, each class of studies has its own advantages and drawbacks. More precisely, the centralized and hierarchical methods provide fully functional resource discovery algorithms but they are badly adapted to the large scale and dynamic nature of the grid environments since the indexing mechanisms for the resource discovery process are centralized. On the other hand, in P2P based systems, the algorithms benefit all the advantages of the underlying environment. However, the methods suffer from the scalability issues, single point of failures or limitations that came with the type of the chosen P2P structure. Lastly, in agent-based studies, the algorithms exhibit the advantages of agent technologies such as flexibility and autonomy. However, the topological structures, which are used by the agents, cause single point of failure and bottleneck problems in large scale and highly dynamic environments such as grids. Resource discovery for query processing in grid environments requires a resource discovery method, which: *i)* is scalable, *ii)* allows discovery of resources by querying more than one resource attribute, *iii)* is prone to node failures and changes in the network topology, *iv)* allows searching resources with attributes within range values and *v)* returns up-to-date results taking dynamic resource attributes into consideration. Considering these requirements, to the best of our knowledge, we cannot find a complete resource discovery method for query processing in grid environments, which suits all listed requirements. For that reason, in this thesis, we first aim at proposing a resource discovery algorithm, which supports all the requirements listed above. In this manner, we first propose self-stabilizing spanning tree construction algorithms, to be used as topology control mechanisms over the grid environment by generating tree structures, which allow scalable and efficient broadcasting

and converge-casting of RD messages in the network. The proposed topology control mechanism is designed to be self-stabilizing to cope up with the dynamicity of the environment. Then, using the constructed tree structure, we propose a resource discovery algorithm, which broadcasts resource requirements and converge-casts up-to-date information of resources to the queried node. For this, we provide a detailed literature survey related to both resource discovery algorithms and self-stabilizing spanning tree construction algorithms. Then we propose three self-stabilizing spanning tree algorithms aiming at construction of spanning trees rooted round the center of the graph. We provide complexity analyses of proposed algorithms. We have implemented the proposed algorithms and two similar existing algorithms in the ns2 simulation environment. We compared our algorithms to each other and to the similar existing algorithms in terms of the spanning tree construction performances and discussed their suitability to the examined problem. Then, we propose spanning tree based resource discovery algorithm exploiting the advantages of the constructed spanning tree. We provide complexity analyses of the algorithm and we consolidate our analyses by implementing the algorithm in *ns2* simulation environment. We compared our algorithm with similar existing studies which use flooding based resource discovery methods and showed weaknesses and strengths of our algorithm.

1.3 Resource Allocation (RA) for Query Processing in Grid Environments

After completion of RD phase, a set of resource nodes, which are capable of executing the query, is returned to the system. From this point, in order to run the tasks effectively, a resource allocation mechanism must be executed in order to: *(i)* choose how many of those resources will be used, *(ii)* which nodes should be involved in the execution of the query and *(iii)* how should the system react in case of node failures during execution of the query. These issues may drastically affect the performance of the query execution in grid environments. Resource allocation for query processing can be defined as selecting and

allocating suitable nodes for executing query operators aiming at optimizing the runtime of the query execution. The optimum allocation of resources for query processing in large scale environments is proved to be NP-Complete [WANG90]. For that reason, existing studies search near optimal solutions for the RA problem using heuristic approaches. However, finding near optimal resource allocation alone may not be sufficient for efficiently processing queries in grid environments; defining the policies in case of node failures is also very important and should be included in the resource allocation method. Since grid environments are dynamic, eventual node failures are likely during the execution of queries. These failures might be very costly if the queries are long running and if the system is not designed fault-tolerant. Therefore fault-tolerance can be considered as a must in processing queries in grid environments. There can be found many studies in the current literature which address the problem of resource allocation for query processing in grid systems [GOUNARIS04, GOUNARIS05, SOE05, GOUNARIS06b, SILVA06, BOSE07, KOTOWSKI08, LIU08, GOUNARIS09]. Several survey studies examine and evaluate these studies with classification [COSTA08, EPIMAKHOV11, COKUSLU12]. Although the existing resource allocation studies provide interesting solutions to this problem, to the best of our knowledge none of these studies consider decreasing the scale of the search space for the candidate resources, which is very important for the scalability of the resource allocation phase. Moreover, we have found few studies which focus on the communication costs of the query execution [BOSE07, LIU08]. Another issue is that although many of these studies examine dynamicity of nodes, we find a very limited number of studies [SMITH05, SMITH07, TAYLOR08, BESTEHORN10] that consider dynamicity of nodes in terms of their existence in the grid environment. However, none of them is specialized on processing stateful query operators especially in grid environments. Considering our analysis on the current literature, we believe that there are still open issues that are not mentioned completely in the resource allocation domain for query processing, regarding the grid systems' characteristics. For these reasons, in this thesis, we also propose new algorithms for resource allocation for query processing in grid systems. We

first provide a detailed literature survey about resource allocation algorithms and fault-tolerant query processing algorithms. Then we propose a new resource allocation algorithm for one operator in a query taking proximities of candidates to the data sources into consideration by scaling down the search space for the candidate resources. Then we propose a resource allocation algorithm for processing the entire query consisting of multiple join operators exploiting the proposed algorithm for one join operator. Lastly we propose a resource allocation protocol for fault-tolerant query processing in grid environments. We provide analyses of the proposed algorithms. Then we consolidate the theoretical analyses with the simulation results and comparisons with the similar existing algorithms.

1.4 Experimental Validation

Grid is a respectively new concept in the information society. Although many discrete event simulators for networking can be found in the literature, most of them do not efficiently model grid environments since they do not include computing capabilities of resources in their model. In our case, in order to test and verify our studies, we need either a real grid environment or a suitable simulator. We have examined existing simulators which are suitable for grid applications namely GridSim, OptorSim, OPNET and ns2, we have evaluated them by considering their suitability to our objectives and we have concluded to use two of them in our simulations, GridSim and ns2. A detailed analysis about the aforementioned simulators can be found in [COKUSLU11_b].

GridSim [SULISTIO08] is a discrete event simulator which is aimed at simulating task scheduling policies in grid environments. It models the resources by considering their MIPS rating, number of processing units, loads and their baud rates. The tasks are simulated by considering input data size, length of task in number of instructions and output data size. The resources, tasks and users are connected via simulated network links

which consider baud rate, propagation delay and maximum transmission unit in bytes. OptorSim [BELL03] is another grid simulation environment which simulates architecture of the EU Data Grid [GAGLIARDI02]. The simulation was constructed assuming that the Grid consists of several sites, each of which may provide computational and data-storage resources for submitted jobs. Computing Elements (CEs) run jobs, which use the data in files stored on Storage Elements (SEs) and a single Resource Broker controls the scheduling of jobs to CEs. In OptorSim, the resources are specified by their number of worker elements in the computing element in each site, number of storage elements in each site, the size of storage elements and bandwidth between each site. The jobs are specified by their file requirements, required file sizes, computing elements' preferences to execute the job and job selection probabilities. OPNET is a commercial high level event based network simulator. It is a very large and powerful network simulator with variety of possibilities such as simulating entire heterogeneous networks with various protocols. OPNET provides various tools for simulation including network model editor, node model editor and process model editor. OPNET supports grid computing for distributed simulations. It provides various computing and networking resource types from various vendors. Therefore it gives the user the ability to generate heterogeneous realistic environments with detailed characteristics of resources. In the application level, it allows users to implement their own jobs to be executed on the simulated environment. Ns2 (Network Simulator 2, version 2.34) [FALL10] is a discrete event simulator which is developed at ISI, California. Ns2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. Ns2 began its development in 1989 as a variant of the REAL network simulator. In years it has evolved substantially and has included contributions from other researchers, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems. The ns2 has become the de facto standard simulator in experimenting wired and wireless network applications since it is an open source and powerful tool in simulating networks.

Although all these simulators provide powerful tools for simulating grid applications,

each of them has some drawbacks that can be used as a decisive factor. For instance, OPNET is a complete simulator, however it is not an open-source software. This property limits the reachability of the environment. On the other hand, OptorSim simulates the grid environment without having support for real implementation of testing protocols. Regarding the characteristics of those simulation environments, we decided that GridSim and ns2 are two suitable simulation environments for validating our algorithms.

1.5 Contributions

In this thesis, we propose algorithms for resource discovery and allocation for query processing in grid environments. We have listed the distinguishing characteristics of grid systems and we aim at contributing the research community by tackling with each one of these characteristics in both problems, RD and RA. In RD part, the proposed algorithms and their contributions are listed as follows:

i. *Self-stabilizing spanning tree construction algorithms:* We propose three self-stabilizing spanning tree algorithms. With these algorithms, we contribute the scalability problem by finding spanning trees with smaller diameters. We also deal with the dynamicity problem by designing our algorithms in a self-stabilizing manner. We compared our algorithms with the similar studies and showed that the proposed algorithms outperform the compared studies in terms of both spanning tree construction costs and spanning tree diameters.

ii. *Spanning Tree Based Resource Discovery (STRD) algorithm:* We propose a spanning tree based resource discovery algorithm, which extracts up-to-date resource information by exploiting the spanning tree. With this algorithm, we tackle the heterogeneity problem by contacting each node in the grid distinctly. We also contribute to the scalability problem during the resource discovery by exploiting multi-cast and converge-cast operations over the spanning tree structure.

In RA problem, the proposed algorithms and their contributions are as follows:

i. *Single Join Operator Resource Allocation (SJORA) algorithm:* In SJORA algorithm, we allocate resources for queries consisting of a single join operator. The SJORA algorithm is composed of the following two consecutive algorithms:

Proximity Based Candidate List Generation (PBCG) algorithm: In this algorithm, we decrease the search space for the candidate list of join tasks. The limited number of candidate resources contributes the scalability of the resource allocation algorithm. We also generate a list of candidate nodes that are close to the data sources. This idea contributes to the performance of the query execution by decreasing the communication costs.

Join Task Resource Allocation (JTRA) algorithm: This algorithm finds optimum allocation of resources for one join task in a query. In this algorithm we deal with the heterogeneity problem by considering different communication costs between candidate and data sources.

ii. *Multi-join Resource Allocation (MJORA) algorithm:* We propose the query resource allocation algorithm to materialize initial resource allocation for a multi-join operator queries. We contribute to the scalability and heterogeneity problems by exploiting the *JTRA* algorithm.

iii. *Fault-Tolerant Resource Allocation (FTRA) Algorithm:* We propose the FTRA algorithm to ensure fault-tolerance during the execution of the query. In this manner, we contribute to the dynamicity problem by passive replication of stateful operators in the query.

1.6 Thesis Organization

The rest of this document is organized as follows: Chapter 2 gives a detailed synthesis of recent studies related to resource discovery and resource allocation methods for query processing in grid systems by introducing classifications and evaluation criteria. We analyze existing studies and discuss their advantages and drawbacks. We identify open issues and potential contributions for RD and RA problems. In Chapter 3, we concentrate

on the resource discovery problem. We first propose self-stabilizing spanning tree algorithms to ensure efficient topology control for resource discovery. Then, we propose a spanning tree based resource discovery algorithm. In Chapter 4, we propose initial resource allocation algorithms for query processing in grid environments. We examine single join operator resource allocation algorithm, and multi-join resource allocation algorithm in detail. The fault-tolerant resource allocation algorithm in grid systems is proposed in Chapter 5. Finally conclusions and perspectives are presented in Chapter 6.

CHAPTER 2: STATE OF THE ART

Résumé

Dans ce chapitre, nous présentons une étude détaillée de la littérature à propos de la découverte de ressources et des algorithmes d'allocation de ressources pour le traitement des requêtes dans les systèmes Grille. En premier lieu, nous examinons les algorithmes de découverte de ressources par classification. En ce qui concerne nos analyses sur les algorithmes de découverte de ressources, nous étudions également les algorithmes auto-stabilisants de construction des arbres couvrants déjà existants et habituellement utilisés comme des mécanismes de contrôle de topologie pour la découverte de ressources. Ensuite, nous passons en revue les algorithmes existants d'allocation des ressources pour le traitement des requêtes dans les systèmes Grille par classification. Enfin, nous examinons les algorithmes à tolérance de panne d'allocation de ressources.

Abstract

In this chapter, we provide a detailed literature survey about resource discovery and resource allocation algorithms for query processing in grid systems. We first examine resource discovery algorithms by classification. Regarding our analyses about the existing resource discovery algorithms, we also survey existing self-stabilizing spanning tree construction algorithms to be used as topology control mechanisms for the resource discovery. Then we survey the existing resource allocation algorithms for query processing in grid systems by classification. Lastly we examine existing fault-tolerant resource allocation algorithms.

2.1 Introduction

Grid systems are very useful platforms for distributed computing, especially for situations in which the scale of data and user requests is very high. They have gained remarkable importance in the last decade since resource requirements of recent

applications increased drastically. Their powerful computing capabilities attract many researchers' attention and lead them to port their research to the grid environments. Distributed query processing is one of these domains in which many studies exist to deal with fundamental characteristics of grid environments such as large scale, heterogeneity and dynamicity. However, these characteristics bring new problems to the query processing domain such as resource discovery, resource selection, resource allocation, autonomous computing, monitoring, replication and caching, security issues and many others [GOUNARIS05]. In this thesis, we are focused on the resource discovery and resource allocation problems for query processing in grid environments. In order to understand the state of the art in these problems, in this chapter, we provide a detailed literature survey related to the resource discovery and resource allocation methods for query processing in grid environments. Firstly, in Section 2.2, we provide analyses for existing resource discovery studies with classification. Then, according to our conclusions on the current resource discovery methods, we directed our attention to topology control algorithms for resource discovery in grid systems. For that, we examined self-stabilizing spanning tree algorithms, which address scalability and dynamicity issues in grid environments. In Section 2.3, we provide a detailed literature survey on resource allocation algorithms for query processing in grid systems. Then, considering the dynamicity characteristic of grid environments, we provide a survey study related to fault-tolerance for query processing in grid. Finally in Section 2.4, we present our conclusions and perspectives about open issues and potential contributions on the RD and RA problems.

2.2 Resource Discovery (RD) for Query Processing in Grid Environments

Resource discovery problem in grid systems can be defined as searching and locating resource candidates, which are suitable for executing jobs in a reasonable time in spite of the dynamicity and large scale of the environment. Success of Grid systems mainly relies on efficient usage of the right resources. Therefore, resource discovery is an important step in finding these resources. But the characteristics of the grid systems make the resource

discovery a time consuming process, which can decrease the performance of the whole system. Several methods have been proposed to solve the resource discovery problem in Grid systems. The existing studies are classified into three main classes in the current literature [COKUSLU09, COKUSLU10_iji, HAMEURLAIN08, HAMEURLAIN10]: methods based on (*i*) centralized and hierarchical systems, (*ii*) P2P systems and (*iii*) agent systems.

In this section, we provide a literature survey related to the resource discovery methods for query processing in grid systems. We examined the existing studies in three classes namely: grid resource discovery methods based on (*i*) centralized and hierarchical architectures, (*ii*) P2P systems and (*iii*) agent systems. A summary of the classification for resource discovery is shown in Figure 2.1. For each class of methods, we describe synthetically the main approach, followed by a deep analysis and comparison with respect to the most important criteria, namely: complexity, scalability, dynamicity, reliability and support for multi-attribute, dynamic-attribute and range queries. The importance and impact of these criteria are explained below:

- Complexity is a basic measure which helps determining the runtime of the algorithm. In our thesis, complexity measure is considered in two aspects, message and time complexities. The message complexity deals with the number of transferred messages. Relatively higher message complexities may result in congestion in the network which may negatively affect the performance of the algorithms. On the other hand, time complexity determines how many steps are required for the termination of the algorithm.
- Scalability is a very important measure, because grids are large scale environments in their nature. The performance of a system which is not scalable, degrades very rapidly as the size of the environment grows. This fact may cause the algorithm to perform poorly in such environments.
- Dynamicity is another important factor in analyzing Grid algorithms since nodes in

Grid systems might be highly dynamic in terms of joining and leaving the system, mostly without any notice. The algorithms that tolerate the dynamicity of the environment are more suitable for grid systems.

- Reliability is also an important measure because in some cases, erroneous query results may cause irrecoverable faults. For instance, RD algorithms, which may result false-positive errors might not be suitable in Grid systems.
- Support for multi-attribute, dynamic-attribute and range queries is a decisive criterion on selecting the methodology in most cases since the running applications may require those types of queries.

We also present a brief literature survey related to self-stabilizing spanning tree construction algorithms, which can be used as topology control mechanisms for resource discovery.

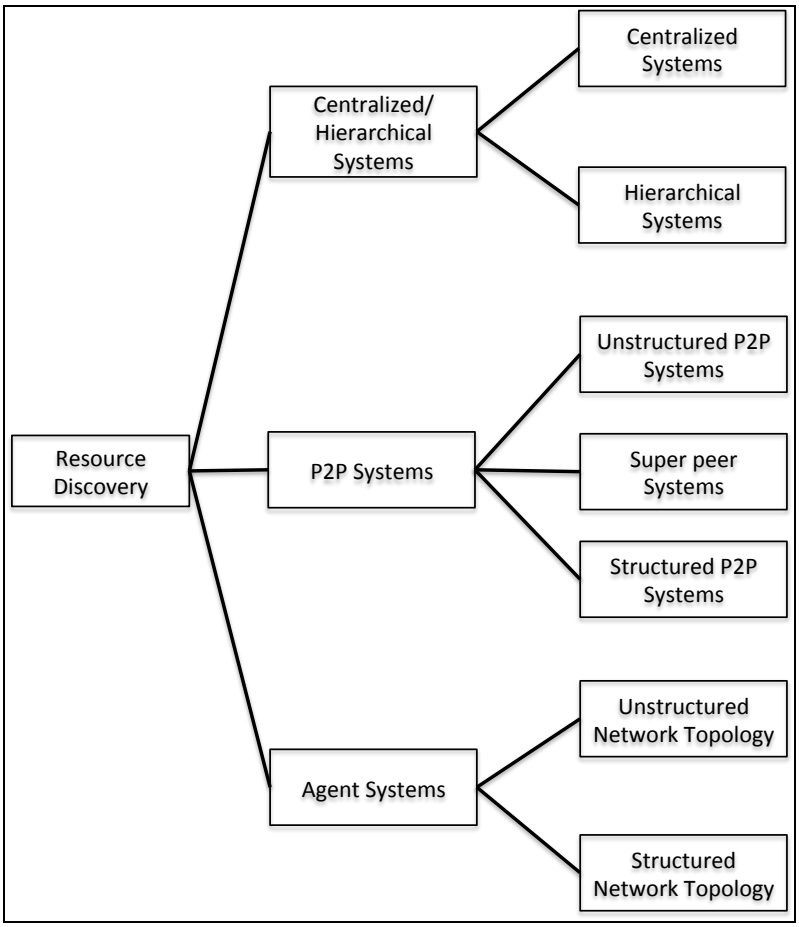

Figure 2.1 Classification for Resource Discovery Algorithms for Query Processing in Grid Environments

2.2.1 Grid Resource Discovery Based on Centralized and Hierarchical Architectures

Centralized and hierarchical systems emerged as suitable approaches, which provide easy to access tools for grid services [ANTONIOLETTI05, ELMROTH05, KAUR07, MOLTO07, RAMOS06, YU03]. In such systems, resource information is stored and updated in central or hierarchically located servers, and resource discovery is realized by querying these servers. Centralized and hierarchical resource discovery mechanisms are mostly used by the web service based grid resource management tools [ANTONIOLETTI05]. There are only few web services based tools that use other classes of approaches such as peer-to-peer resource discovery [TALIA04, TALIA05]. In this

subsection, we focus on centralized and hierarchical systems and present a survey of recent studies, which provide grid resource discovery services using centralized and hierarchical approaches [BEHESHTI07, CZAJKOWSKI01, ELMROTH05, FITZGERALD97, FOSTER97, KAUR07, LI02, MOLTO07, RAMOS06, RIEDEL07, YU03]. We classify them according to the topology of resource information. In centralized methods, resource information is stored in a central server, whereas in hierarchical methods resource information is divided and partially distributed to several locations. We give detailed analysis for each study and evaluate them according to important qualitative criteria namely, scalability, dynamicity, reliability and support for multi-attribute, dynamic-attribute and range queries.

2.2.1.1 Grid Resource Discovery Using Centralized Systems

Grid resource discovery using centralized systems is excessively studied and implemented in grid environments. The main idea behind this class of methods relies on the central management of metadata related to the resource information in the grid environment. Most of the studies in this class profit from existing information provider systems such as LDAP (Lightweight Directory Access Protocol) or UDDI (Universal Description Discovery and Integration). There can be found many studies in this class, which became very popular in the first decades of grid systems [FITZGERALD97, KAUR07, MOLTO07, RIEDEL07, YU03, MOLTO07]. For instance, in [YU03], Yu et al. developed a web services based grid service publication directory system (GMD) in which service for resources and clients are provided via web by using XML formatted messages. Kaur and Sengupta [KAUR07] presented a centralized resource discovery mechanism for grids which relies on web services. In [MOLTO07], Molto et al. proposed a metascheduler grid service that can be accessed through the network by users who are interested in task allocation and scheduling in computational grids. In [RIEDEL07], authors present a web services based grid middleware (UNICORE 6) in which resources are represented by web

services. In [FITZGERALD97], authors proposed a Meta-computing Directory Service (MDS) for resource management in grid systems which is based on LDAP central servers.

The grid resource discovery using centralized systems provide grid middleware developers an easy to use interface to manage grid resources. They keep grid resource information by using centralized databases. In a large-scale grid environment, the centralization of the service may easily create bottlenecks on the central servers. The bottleneck problem may be raised both because of frequent resource updates or large number of query requests waiting to be processed. The centralization causes another important problem in dynamic grids as being a single point of failure. Failure of one of the central servers in the system may cause the whole system to become unavailable. In some approaches, the idea of replication of central servers is depicted in order to eliminate single point of failures. But replication of servers in a large-scale dynamic grid may be very expensive in terms of communication costs. The proposed systems support the multi-attribute and range queries since the resource information is stored in databases, which are capable of processing complex queries. But since the update of dynamic resource attributes are held in discrete intervals, most of these systems do not support dynamic-attribute queries.

2.2.1.2 Grid Resource Discovery Using Hierarchical Systems

In recent years, as the size of the grid environments grows, researchers directed their attention to hierarchical systems to overcome the problems caused by centralized systems in resource discovery. Many algorithms are proposed in this class to address drawbacks of the centralized systems [BEHESHTI07, ELMROTH05, LI02, RAMOS06, YIN07]. From those, Elmroth and Tordsson proposed a grid resource broker and job submission system in [ELMROTH05] in which index servers, which are responsible for resource information, are organized in a hierarchical topology. In [RAMOS06], Ramos et al. proposed a web service for resource discovery in grids, based on Globus Toolkit (GT3), a hierarchical topology in which the grid environment is divided into virtual organizations (VO).

Beheshti and Moshkenani proposed a resource discovery method by joining agents, ontologies and web services [BEHESHTI07]. They used SOA concept in which the metadata related to resources is distributed to a portion of nodes. In [LI02] Li et al. presented a grid resource discovery model which is inspired by network routing mechanisms. In [YIN07], Yin et al. proposed a 3-layer hierarchical grid resource discovery method.

The hierarchical systems based grid resource discovery algorithms provide more scalable platforms than the centralized ones and still provide simple user interfaces to manage grid resources. In a large-scale grid environment, the hierarchical topology of the service decreases the probability of bottleneck problem. However single point of failure problem still exists since failure of one of the master servers in the system may cause a large part of the nodes become invisible to the queries. All proposed algorithms in this class support the multi-attribute and range queries since the resource information is stored in databases that are capable of processing complex queries. And since, in many studies, the information is verified in the resource nodes after querying databases, the proposed algorithms also support dynamic-attribute queries.

2.2.1.3 Comparison

The summary of comparison between resource discovery using centralized and hierarchical systems can be seen in Table 2.1.

	RD Using Centralized Systems	**RD Using Hierarchical Systems**
Scalability	Not scalable due to bottleneck problem	Better scalable because of the hierarchical distribution of load
Dynamicity	Tolerant to node dynamicity, but not tolerant to indexing mechanism's dynamicity	Tolerant to node dynamicity, better tolerant to indexing mechanism's dynamicity
Reliability	Reliable in terms of query correctness, but not reliable in terms	Reliable in terms of query correctness, better reliable in terms of single point of

	of single point of failure	failure
Range Queries	Supported since queries are resolved in centralized servers without any hashing	Supported since queries are resolved within hierarchically distributed servers without any hashing
Multi-attribute Queries	Supported since queries are resolved in centralized servers without any hashing	Supported since queries are resolved within the super-peers without any hashing
Dynamic-attribute Queries	Not supported because of the periodic updates	Not supported because of the periodic updates

Table 2.1 Comparison summary between two resource discovery methods

Both centralized and hierarchical methods behave closely against nearly all the evaluation criteria. The main difference is in scalability and reliability. The centralized systems suffer from the bottleneck problems in large scale. There also exists the single point of failure problem. Even though some studies propose to replicate the centralized index server, this procedure might be very expensive in terms of messaging complexity in large scale. On the other hand, hierarchical systems distribute the load into many locations instead of one central server. This property increases the scalability of the system by distributing load on index servers. They also decrease the effect of single point of failures. In case of a failure of an index server, a part of the system becomes unreachable instead of the whole. Support for dynamic attribute queries requires that the query be processed within the resource nodes. Since the idea in the examined solutions is indexing the resource information in central locations, they do not support dynamic attribute queries. Some of the examined algorithms propose solutions for this problem, but the solutions are independent from the classification that we propose. By taking the evaluations into consideration, we can say that the centralized methods are not suitable for the large-scale environments. But they might be well suited to the systems in which the scale is small and indexing server is reliable. In such cases centralized systems can be used effectively. On the other hand, hierarchical methods are more suitable for environments in which scale is bigger since the load is distributed to many locations. But even the load is hierarchically distributed; those

methods may still suffer from bottleneck problem in large scale.

2.2.1.4 Conclusions

In this section, we have synthesized and analyzed recent grid resource discovery methods, which are based on centralized and hierarchical systems. We evaluated them according to the aforementioned qualitative criteria, and compared different classes of methods with each other. These types of resource discovery algorithms seem to have many disadvantages in large-scale dynamic grid environments. But their simple design brings them to the foreground and makes them suitable techniques to be used in many grid management tools. Most web service based grid management tools use centralized and hierarchical grid resource discovery techniques. With regards to the examined methods, it can be said that centralized and hierarchical systems based approaches are suitable for small-scale grid environments in which the dynamicity of nodes is low. In such environments, they provide a very simple and standard resource management platform. There is vast amount of ongoing research in this topic, which is aimed at solving problems mostly caused by scalability issues.

2.2.2 Grid Resource Discovery Based on Peer to Peer (P2P) Systems

In the first decade of Grid systems, centralized and hierarchical systems emerged as suitable approaches that provide an easy to use platform independent tool for Grid services [ANTONIOLETTI05]. However, most of these methods could not be adapted to today's large scale environments since scalability and dynamicity in grids restrict their usage area. For these reasons, other approaches were investigated to be used in grid systems to overcome these problems. Thus, researchers directed their attention to the P2P systems to evaluate their capabilities on grid platforms. Synergy and convergence between grid and P2P systems were clearly pointed out in [IAMNITCHI03, IAMNITCHI05, TALIA05]. P2P systems were proven to work efficiently under large-scale environments. A rich

survey on P2P systems can be found in [ANDROUTSELLIS04]. The P2P based algorithms can be classified into three classes depending on how peers are organized: *i)* structured P2P systems, *ii)* unstructured P2P systems and *iii)* super-peer systems [HAMEURLAIN10]. In unstructured P2P systems, the peers are not organized to construct any topology and resource discovery is realized by diffusion of messages to the network. In super-peer systems, some nodes are selected to act as directory services (super-peers). Resource information is held by those super-peers in order to find resources in the grid system. In structured P2P systems, peers are organized into virtual organizations. Discovery of resources in such systems is realized by using distributed hash tables (DHT). Different methods were proposed by using all these different P2P systems [CAI03, CHEEMA05, MARZOLLA07, TRUNFIO07]. Very comprehensive and detailed survey studies on the grid resource discovery based on P2P systems can be found in [TRUNFIO07, RANJAN08, SEDAGHAT08, HAMEURLAIN10].

In this section, we first describe recent resource discovery algorithms based on three different classes of P2P systems: Unstructured P2P Systems, Super-peer Systems and Structured P2P Systems. Next, we give a detailed analysis of these algorithms. Then we provide a qualitative evaluation for each class of algorithms with respect to the introduced criteria.

2.2.2.1 Grid Resource Discovery Based on Unstructured P2P Systems

The unstructured P2P systems are the first milestones of the P2P systems. The participants do not construct a special network topology. Instead, they are connected to each other through their neighboring nodes. In unstructured P2P systems based grid resource discovery studies [FILALI08, IAMNITCHI03], each node in the grid is treated as a participant of the P2P system and is responsible for providing its resource information. For instance, Iamnitchi and Foster [IAMNITCHI03] proposed a P2P approach in which nodes construct an unstructured P2P system by publishing their resource information to the network. Filali, Huet, and Vergoni [FILALI08] proposed an unstructured P2P based

resource discovery method for Grids in which the resource nodes send their resource information periodically to their neighbors and the neighbors store this information in their cache.

Considering the nature of the unstructured P2P resource discovery techniques, in most cases, because of the common routing mechanisms; the complexity of the algorithms are around $O(N^2)$ which makes the approach unscalable. The message and in some cases time complexities have higher order of growth than the scale of the network.

On the other hand, this approach can easily handle dynamicity of the Grid since both resources and indexing nodes are distributed to the entire network. In any case, even if the network is very dynamic, queries are not lost in the network and propagation of the queries continues until a TTL value is reached. Nearly all unstructured systems suffer from false-positive errors caused by the usage of TTL limitations. Even if the searched resources exist and are available on the Grid, the system may return unsuccessful results to the queries because the TTL limit is reached. Otherwise, when TTL is set to a higher value, asymptotic increase of the messages negatively affects the bandwidth and runtime of the algorithms.

Nevertheless, the unstructured P2P systems support range, multi-attribute and dynamic-attribute queries easily.

2.2.2.2 Grid Resource Discovery Based on Super-Peer Systems

The super-peer systems emerged as an alternative to the unstructured P2P systems in order to address scalability issues. In super-peer systems, some nodes (super-peers) in the environment are considered to be privileged and act differently from the remaining nodes [YANG03]. Most commonly, in super-peer based resource discovery algorithms, the super-peers are responsible for keeping the information of a group of nodes. There can be found many studies in the current literature which use super-peer systems based resource discovery [MARZOLLA05, MASTROIANNI05, PUPPIN05]. Mastroianni, Talia, and Verta [MASTROIANNI05] proposed a resource discovery mechanism in which some nodes are selected as super-peers acting as directory services. Puppin et al. [PUPPIN05]

proposed a grid information service based on super-peer approach. They defined some nodes as super-peers, and then created clusters by using the super-peer neighborhoods. Marzolla, Mordacchini, and Orlando [MARZOLLA05] defined the concept of Workload Management Systems (WMS) which act as an indexing service for a subset of virtual organizations in the Grid.

Most super-peer based P2P algorithms use flooding between the super-peers. Decreasing the size of the flooding domain reduces time and message complexities of algorithms. Nearly all of this type of algorithms have message complexities $O(S^2)$ and time complexities $O(S)$ where S is the number of super-peers in the network. Even these types of algorithms can be considered as more scalable than unstructured systems; super-peers may suffer from being bottlenecks in the system when the number of requests is large.

Moreover, the super-peers are responsible for a set of resources and failure of a super-peer will break the imaginary connection of the resources which exist and are available. Therefore, dynamicity of a super-peer badly affects the domain of the queries. This fact also negatively affects the reliability of this approach by turning super-peers into single point of failures. However, since the queries are resolved by super-peers by checking the index tables, this approach supports range queries and multi-attribute queries easily. But because of the resource information is collected by the super-peers at periodic intervals, this method does not support dynamic-attribute queries in its nature.

2.2.2.3 Grid Resource Discovery Based on Structured P2P Systems

The structured P2P systems become popular in the last decades of the P2P systems by the use of distributed hash tables (DHT). In structured P2P systems, the identifier values of information is hashed and mapped to a set of nodes in the system in a deterministic or distributed fashion [ELANSARY03]. The structured P2P systems based resource discovery methods [ANDRZEJAK02, CAI03, OPPENHEIMER04, SPENCE03, TALIA07] use DHT's to store resources' information. Therefore the resource discovery is realized by DHT lookups in most of these studies. For instance, Cai et al. [CAI03] proposed a grid

resource discovery system (MAAN), based on P2P, which supports multi attribute and range queries. Each MAAN node is an instance of a Chord system and the resource information is mapped to Chord key-space. Andrzejak [ANDRZEJAK02] proposed a P2P grid resource discovery mechanism based on CAN P2P system. Oppenheimer et al. [OPPENHEIMER04] proposed a structured P2P based resource discovery mechanism (SWORD) which supports both range queries and multi-attribute queries. In SWORD, the server nodes are connected to each other by using Bamboo P2P system and use DHT system to index the resources. In XenoSearch, Spence and Harris [SPENCE03] proposed a P2P based resource allocation method for distributed environments. The proposed algorithm is an extension of the Pastry P2P system [ROWSTRON01]. Talia, Trunfio, and Zeng [TALIA07] proposed a DHT based resource discovery mechanism for large scale Grids which is based on Chord Structured DHT system.

Since these algorithms use topological structures, time and message complexities of the algorithms are around $O(logN)$. In many algorithms all resource nodes get involved in the query processing, which means that, theoretically, all nodes will have the same load. This eliminates the bottlenecks in the system and ensures the scalability of the structured P2P approach. On the other hand, in most algorithms, the queries are distributed to the network by following a defined path in the topological structure. Therefore, failure of a node which will forward the query may result in loss of queries in the network. This brings the single point of failure problem in a dynamic Grid environment. But, the use of structured query routing mechanisms eliminates false-positive errors since the query is relayed to the end of its search domain. Even if the nature of structured P2P based resource discovery algorithms do not support range, multi-attribute and dynamic-attribute queries, nearly all algorithms that are developed in this scope find reasonable solutions to support all different types of queries.

2.2.2.4 Comparison

We described and analyzed several grid resource discovery algorithms that are developed by using different P2P techniques. Although each algorithm has its own advantages and disadvantages, commonalities can be clearly distinguished when they are examined in their own classes. Regarding those commonalities, a comparison can be easily performed between three different classes of P2P techniques. A summary of comparison can be seen in Table 2.2 which is described below.

	Unstructured P2P	**Super-peer P2P**	**Structured P2P**
Scalability	Not scalable due to time and message complexities	Not scalable due to bottlenecks	Scalable since complexities are low and load is distributed
Dynamicity	Tolerant to node dynamicity since queries are resolved within the nodes	Performs poorly when the Grid is dynamic	Performs poorly when the Grid is dynamic
Reliability	Not reliable because of the false-positive errors	Not reliable because of the false-positive errors and single point of failures	Reliable since no single point of failures and no false-positive errors exist
Range Queries	Supported since queries are resolved within the nodes without any hashing	Supported since queries are resolved within the super-peers without any hashing	Supported using complicated techniques
Multi-attribute Queries	Supported since queries are resolved within the nodes without any hashing	Supported since queries are resolved within the super-peers without any hashing	Supported using complicated techniques
Dynamic-attribute Queries	Supported since queries return always up-to-date results	not supported since resource information in the super-peers is updated in discrete	Supported using complicated techniques

		intervals	

Table 2.2 Summary of comparison between P2P based resource discovery methods

Regarding our analyses, we can say that the unstructured P2P based grid resource discovery systems are suitable for small scale, highly dynamic Grid environments in which different types of queries are required. If the scale is large, and false-positive errors are fatal, then other methods should be examined. Super-peer based grid resource discovery mechanisms are suitable for middle-scale Grid networks in which reliability of super-peers is strictly provided. They are not suitable for dynamic-attribute queries and if the false positive-errors cause serious problems. Structured P2P based methods, on the other hand, are suitable for large-scale Grid systems in which reliability is important and dynamicity is low.

2.2.2.5 Conclusions

In this section, we provided a literature survey related to resource discovery methods based on P2P systems with classification. For each class of methods, we described synthetically the main idea, with a detailed analysis. A qualitative evaluation of described methods was provided. With respect to our analyses and evaluations, it can be concluded that P2P systems provide a wide range of solutions to the resource discovery problem meeting different requirements in each class. The unstructured P2P systems based approaches are considered to be the most suitable solutions regarding the reliability, dynamicity and support for different types of queries properties. However they are poorly adapted to the large scale of the environment since the underlying multicasting methods are flooding based in general. On the other hand, super-peer based systems answers the scalability issues with trading off the dynamicity and reliability problems. The most recent solutions, structured P2P systems based resource discovery, conforms many of the required specifications by trading of simplicity of the RD method. We believe that P2P systems provide very suitable platforms to be used in the RD problem. Advantages of each class should be examined carefully and a combination of them should be studied as an

alternative RD method.

2.2.3 Grid Resource Discovery Based on Agent Technologies

In this section, we propose a synthetic review of the state of the art and analysis of some recent resource discovery algorithms that are based on agent systems. Many of these algorithms profit from using agents as monitoring services. On the other hand studies, which are based on mobile agents (MA), profit mainly from their autonomy property, which allows the query to determine migration site by itself. Although the type of the utilized agent differs from each other, we decided to classify them according to their underlying network topologies since we believe that underlying topology has more impact on the evaluation of these algorithms. In this perspective, we defined two classes: algorithms that do not rely on a structured network topology and those that generate a network topology to accomplish resource discovery tasks. We also provide an evaluation for each class of grid resource discovery algorithms. Then, we show the comparison between RD methods based on different classes of agent systems.

2.2.3.1 Agent Based Grid Resource Discovery on Unstructured Network Topology

In agent based grid resource discovery approaches in which the underlying network topology is unstructured [DING05, JUN00, TANG06, YU06], the agents and the agent requests are diffused to the environment freely without any pattern to search resources. More precisely, Ding et al. [DING05] proposed a heuristic agent-based resource discovery algorithm in their study in which agents cooperate to find available resources. Jun et al. [JUN00] introduced an agent-based resource discovery model in which the agents running at different nodes learn about the existence of each other using a mechanism called distributed awareness. In [YU06], a mobile agent based grid resource management system is presented in which an information server creates mobile agents according to the required resources' properties. Tang and Huang [TANG06] proposed a grid resource management

algorithm based on mobile agents. They also described an architectural model for acquisition of Grid information.

In agent based grid resource discovery approaches in which the underlying network topology is unstructured, the system does not suffer from bottleneck problem. The main factor which affects the scalability of the system is diffusion technique of the requests. When agents are used, the diffusion is handled by using flooding approach which is unscalable because of its message complexity. On the other hand, if mobile agents are used, because of their autonomy and self-decision properties, more clever routing techniques are applied to increase scalability. On the other hand, when the Grid is dynamic, since in agent based approaches flooding is used to distribute the queries, dynamicity of the nodes does not perturb the dissemination of queries. But when the mobile agents are used, the queries are routed on a single path, and the failure on any node on this path may cause loss of queries in the network. The algorithms do not have single point of failures in general since central managers do not exist. Moreover, the distribution of queries is not limited by a TTL value which eliminates false-positive errors. But we believe that in large scale environments in which an unstructured network topology exists, TTL values are essential to avoid extremely long query response durations. Nevertheless, nearly all analyzed algorithms which belong to this classification support range, multi-attribute and dynamic-attribute queries easily since they process queries within the nodes without any discrete mapping function such as hashing.

2.2.3.2 Agent Based Grid Resource Discovery on Structured Network Topology

In this class of studies [KAKARONTZAS06, MANVI05, PUH07, YAN07], the agents generate structures such as clustered virtual organizations to hierarchically distribute resource information management. The resource discovery queries are processed by agents that are responsible from a group of resources' information. For instance, Yan et al. [YAN07] proposed a system in which the resources are divided into some Virtual Organizations (VO) each having an index server and several nodes. Kakarontzas and

Savvas [KAKARONTZAS06] presented an agent-based approach to grid resource discovery. The approach is based on client agents which act on behalf of Grid users. In [MANVI05], an agent based resource allocation model (ARAM) is presented in which the resource information is updated by interacting with the grid information servers. In their design, mobile agents, which are called Job Agents (JA), search for the available resources. Puh, Jezic, and Kusek [PUH07] proposed a multi-agent resource discovery algorithm for Grids.

In structured networks, the bottleneck problem is eliminated since the load on nodes is distributed. Moreover, because of the structured nature, migration of mobile agents or distribution of queries is held with more efficient routing techniques which increase the scalability of the system. But this advantage brings some problems such as single point of failures. When the Grid is dynamic, since some nodes act as relay nodes or manager of a subset of resource nodes, the dynamicity of the network may perturb the dissemination of queries. The failure of a node in the structure will cause loss of queries in the network. Moreover, failure of a node which manages a large subset of resources will cause a large number of resources to become invisible to the queries even if they exist. On the other hand, nearly all analyzed algorithms which belong to this class support range and multi-attribute queries easily since they are processed within the nodes without any discrete mapping function such as hashing. The dynamic-attribute queries are also supported by the use of agent technology which allows the resources to send updates about their dynamic attributes continuously instead of at periodical intervals.

2.2.3.3 Comparison

A summary of the comparison can be seen in Table 2.3. The agent based resource discovery systems that use unstructured network topology is less scalable than those which use structured network topology because of the larger time and message complexities. However they are more tolerant to dynamicity of environment caused by the use of flooding of messages. Both classes of methods are considered to be unreliable because of

either single point of failures or false-positive errors. Luckily both classes of methods support all types of queries.

	Unstructured Network Topology	**Structured Network Topology**
Scalability	Not scalable due to time and message complexities	Scalable because of the hierarchical distribution of load
Dynamicity	Tolerant to node dynamicity since queries are distributed in parallel, but not tolerant in some approaches in which query migrates on a single path	Not tolerant since dynamicity of nodes in the structure may result in disconnectivity of a large portion of the resources
Reliability	Not reliable because of either false-positive errors or single point of failures	Not reliable because of the single point of failures
Range Queries	Supported since queries are resolved within the resource nodes without any hashing	Supported since queries are resolved within the nodes without any hashing
Multi-attribute Queries	Supported since queries are resolved within the resource nodes without any hashing	Supported since queries are resolved within the nodes without any hashing
Dynamic-attribute Queries	Supported since queries are resolved within the resource nodes without any hashing	Supported since agents update the resource information dynamically

Table 2.3 Summary of comparison between agent based resource discovery methods

2.2.3.4 Conclusions

Several agent based grid resource discovery algorithms are described and analyzed in this section. Comparison of each class of agent based resource discovery methods is

explained with respect to the defined criteria. Both structured and unstructured network topology based methods have their own advantages and disadvantages. Regarding the analysis, we can say that on top of an unstructured network topology, scalability and dynamicity properties depend on which type of agent is used: mobile agent or agent. In large scale grids in which dynamicity of nodes is relatively low, mobile agent based systems are preferable because of their autonomy properties that allow efficient migration. If the scale is relatively small and dynamicity is high, agent based algorithms become advantageous since they eliminate single point of failures. On the other hand in structured network topologies, regardless of which kind of agent is used, the system is scalable since hierarchical structuring of the network decreases the complexity without generating bottlenecks. But those systems generally suffer from the single point of failures since manager nodes' dynamicity badly affects the reliability.

2.2.4 Global Evaluation of Resource Discovery Methods

We analyzed and evaluated recent studies related to different methodologies used in grid resource discovery in the previous sections. We focused on centralized/hierarchical systems, P2P systems and agent systems based grid resource discovery techniques for query processing. In Table 2.4, a summary of comparison of the different methodologies can be found.

	RD Based on Centralized / Hierarchical Systems	**RD Based on P2P Systems**	**RD Based on Agent Systems**
Scalability	Suitable for small-scale grid environments	Suitable for large-scale grid environments	Suitable for small-medium scale grid environments
Dynamicity	Tolerant to resource dynamicity but not tolerant to indexing mechanism's	Perform poorly when the environment is dynamic	Tolerant because of the autonomy property of agents

	dynamicity		
Reliability	Reliable in terms of query correctness, but not reliable in terms of single point of failure	Reliable since false-positive errors and single point of failures do not exist	Not reliable because of either false-positive errors or single point of failures
Range Queries	Supported	Supported using complicated techniques	Supported since queries are resolved within the resource nodes without any hashing
Multi-attribute Queries	Supported	Supported using complicated techniques	Supported since queries are resolved within the resource nodes without any hashing
Dynamic-attribute Queries	Not supported because of periodic updates of resource information	Supported using complicated techniques	Supported since queries are resolved within the resource nodes without any hashing

Table 2.4 Summary of Comparison Between Different Classes of RD Methods

The Centralized/Hierarchical systems, P2P systems and Agent systems based techniques are broadly being studied in today's grid resource discovery systems. The resource discovery methods based on the Centralized and Hierarchical systems were heavily used in the first decade of grid environments. They became a de-facto standard in grid resource management very rapidly [ANTONIOLETTI05]. Their simple design brings them to the foreground and makes them suitable techniques to be used in many grid management tools. However, these methods have many disadvantages in today's large scale dynamic grid environments. With regards to the examined studies, we can say that these methods are suitable for small scale grid environments in which the dynamicity of nodes is low. In such environments, they provide a very simple and standard resource management platform. There is vast amount of ongoing research in this topic which is aimed at solving problems mostly caused by the scalability issues. On the other side, most recent P2P techniques use

structured DHT systems which increase the performance of the queries drastically. Moreover, usage of the DHT mappings brings the scalability and reliability of the P2P systems since all nodes in the system involves the resource discovery process. But, DHTs are not miraculous and have some disadvantages for the resource discovery domain. The usage of DHTs limits the RD algorithm in terms of support for dynamic-attribute queries. Since dynamic-attributes of resources are changing in time, keeping these attributes in DHTs is not feasible. To solve this problem, many algorithms use the topological structure of the overlay to efficiently distribute the query directly between the resource nodes. Inheriting all properties of overlay systems, P2P based grid resource discovery methods are suitable for large scale dynamic environments in which reliability of queries is important. Lastly, agent based systems are attractive, mainly because of their autonomy property. They have capabilities to determine new migration sites according to their migration policies. This property might easily be used to accomplish efficient route selection for queries which converges to the result in each step. However, their non-deterministic nature results in false positive errors in many recent studies in which inefficient flooding techniques are avoided. Most of the agent based RD techniques process the queries inside the resource nodes while the query is traversing the network. This brings the advantage of having up-to-date resource information all the time which makes this class of algorithms support dynamic-attribute queries. Regarding these attributes, agent based resource discovery methods are suitable for dynamic middle-scale Grid environments in which some false-positive errors are acceptable.

2.2.5 Self-stabilizing Spanning Tree Construction Algorithms

Spanning tree algorithms are widely used in many distributed applications as efficient topology control mechanisms. A spanning tree is a subset S of a graph G which contains every node in G and which does not contain any cycles. With the growth in the scale of distributed systems such as grid systems, the need for such topological control mechanisms has gained significant importance. These control mechanisms decrease the complexity of

distributed algorithms caused by the connectivity of the underlying graph. By using spanning trees, many distributed applications, especially those involving multicast or broadcast operations, can be optimized by making use of their properties. Many algorithms have been developed to build different types of spanning trees. Several studies focus on constructing minimum spanning trees in which the sum of edge weights is minimized [AHUJA89, AWERBUCH87, GALLAGHER83, LIEN88]. Studies such as [BLIN09] construct a minimum degree spanning tree in which degrees of vertices are minimized. Some other studies aim at constructing minimum diameter spanning trees in which diameter of the resulting spanning tree is minimized [BUTELLE95]. Besides classical distributed spanning tree construction algorithms, some of these studies also consider dynamicity and fault tolerance in their design. Self-stabilizing spanning tree algorithms have gained importance recently [GARTNER03] by the wide spread usage of unstable systems. Self-stabilizing concept guarantees the validity of spanning tree structure without having the need to regenerate the spanning tree every time the structure has changed [DIJKSTRA74]. The idea of self-stabilizing algorithms is that independent of the global state of the system, the system will reach to a correct global state after a finite amount of time. Self-stabilization is a very useful approach for the systems in which dynamicity occurs frequently, such as grid systems.

2.2.5.1 Recent studies

There can be found many studies in the current literature related to self-stabilizing spanning tree construction [AFEK91, ANTONOIU95, ANTONOIU97, BAALA03, DOLEV93, GUPTA03, HERAULT06, KOSOWSKI06, PAN99]. Afek et al. [AFEK91] proposed a memory-efficient self-stabilizing spanning tree algorithm for general networks. Dolev et al. [DOLEV93] proposed a uniform BFS spanning tree algorithm based on the id's of the nodes. Butelle et al. [BUTELLE95] presented a uniform self-stabilizing algorithm which finds a minimum diameter spanning tree of an arbitrary positively real-weighted graph. Antonoiu and Srimani [ANTONOIU95] proposed a self-stabilizing

algorithm to construct an arbitrary spanning tree assuming the existence of a root node, which is used as a reference node to build the spanning tree. In [ANTONOIU97], Antonoiu and Srimani proposed another self-stabilizing algorithm which constructs minimum spanning trees. Pan et al. [PAN99] proposed a self-stabilizing spanning tree construction algorithm based on self-stabilizing maximum finding method. In their algorithm they find the maximum identifier and determine distances of each node to the maximum identifier node. Baala et al. [BAALA03] presented a random walk based spanning tree construction algorithm which is self-stabilizing. Their algorithm is based on random walk strategy, which is executed by independent mobile agents. Gupta and Srimani proposed two self-stabilizing spanning tree algorithms in [GUPTA03]. They consider ad-hoc networks as the system model in their design. In [HERAULT06], Herault et al. proposed a self-stabilizing spanning tree algorithm for large scale systems in which the biggest id node becomes the root of the final tree. In [KOSOWSKI06], Kosowski and Kuszner proposed two self-stabilizing algorithms to find spanning tree in a polynomial number of rounds based on the ids of the nodes. Blin et al. [BLIN09] proposed a self-stabilizing algorithm to find minimum degree spanning tree in a network. They have extended the study proposed in [AFEK91] by adding a degree reduction module which decreases the degree of the resulting spanning tree in each round.

2.2.5.2 Conclusions

Many of these algorithms construct spanning trees using the unique ids of the nodes in the network [AFEK91, DOLEV93, HERAULT06, KOSOWSKI06, PAN99]. Others use different parameters such as unique weight of edges [ANTONOIU97], assumption of existence of privileged nodes [ANTONOIU95, GUPTA03] or random walks [BAALA03]. Although these spanning tree construction methods provide fruitful solutions to the topology control problem, to the best of our knowledge, we cannot find an algorithm, which aims at finding a spanning tree that optimizes the distribution of messages. We believe that many alternative meaningful metrics, such as degrees of nodes, can be used in

constructing the spanning tree instead of the ids of nodes.

2.2.6 Conclusions for Resource Discovery Algorithms

We believe that combining advantages of different RD approaches would result in useful studies for resource discovery in grids. For example, by using web services, one can provide a simple interface to the users, while for the resource discovery, by using mobile agents or P2P techniques, a scalable, reliable and also easy to use resource discovery middleware can be designed. To the best of our knowledge, we cannot find a research work, which is directly focused on addressing the problems, which are caused by the characteristics of the grid environments and requirements of queries. For this purpose, in this thesis, we propose a new cross-layer design approach in resource discovery, which combines topology control and efficient resource discovery together.

Considering our analyses, we concluded that scalability should be taken into account in the first glance. For this, we figured out that efficient topology control algorithms, which are suitable for dissemination of resource discovery messages, should be studied such as spanning trees. Moreover, for the dynamicity issues, we figured out that the topology control algorithm, which is going to be used in the RD step, should be prone to the dynamicity of the system. For that reasons we directed our attention to the self-stabilizing spanning tree algorithms. We believe that a resource discovery algorithm, which inspires all the advantages of the examined studies with a fault-tolerant topology control mechanism, should be our objective for proposing efficient resource discovery algorithms.

2.3 Resource Allocation (RA) for Query Processing in Grid Environments

Resource Allocation 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 [HAMEURLAIN10], 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 [GOUNARIS04, SLIMANI04, BUYYA05, GOUNARIS05, KANT05, MANDAL05, SOE05, GOUNARIS06b, SILVA06, VENUGOPAL06, BOSE07, ZHAO07, KOTOWSKI08, LIU08, GOUNARIS09, RUIZ09, PATON09]. Existing studies are generally classified into two classes in the current literature: static resource allocation and dynamic resource allocation. In static RA approach, the allocation is performed once, and allocated nodes sustain execution until the tasks are completed. On the other hand, in dynamic RA approach, according to the monitored status of the resources, the allocation is dynamically modified during the execution of the tasks. However, we believe that dynamicity of the RA algorithm should be examined as a design parameter instead of a classification metric. For that reason, in this section we classify the existing RA algorithms based on their most common attributes. 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. The classification is shown in Figure 2.2.

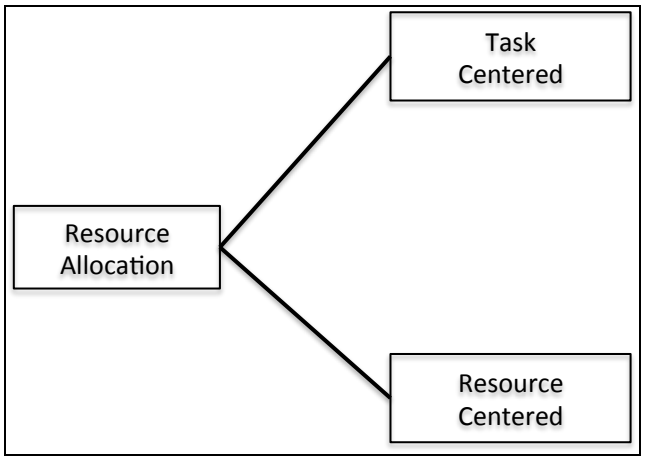

Figure 2.2 Classification for Resource Allocation Algorithms for Query Processing in Grid Environments

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 chapter, 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 that 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 [BACA89, WANG90], therefore in the resource allocation step, 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 [SULISTIO08] 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 section, in Section 2.3.1, we examine recent survey studies in resource

allocation for grid systems. In Section 2.3.2, we provide a detailed survey of resource allocation methods according to our classification. In Section 2.3.3, we present a literature survey related to fault-tolerance for query processing in grid environments in order to address dynamicity of nodes in terms of their existence. Finally, in Section 2.3.4, we discuss our conclusions and perspectives.

2.3.1 Existing Survey Studies and Motivations

Detailed survey studies can be found in the literature related to resource allocation and job scheduling in grid systems [KRAUTER02, JIAN07, JIANG07, COSTA08, EPIMAKHOV11]. Even though in this section 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 [JIAN07, JIANG07, KRAUTER02]. On the other hand, we find very few survey studies that are focused on the query processing domain [COSTA08, EPIMAKHOV11].

In [KRAUTER02], 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 [JIAN07], 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 [JIANG07], 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 [COSTA08], 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 [EPIMAKHOV11], 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 section, 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 present evaluations and comparisons between different classes by both qualitative and quantitative criteria.

2.3.2 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.

2.3.2.1 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. There can be found many studies in this class [BOSE07, GOUNARIS04, GOUNARIS06b, KANT05, LIU08, MANDAL05, SILVA06, VENUGOPAL06]. For instance, in [KANT05], Kant and Grosu proposed auction-based resource allocation protocols. They proposed three auction protocols using parameters characterizing resources and jobs. 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. Mandal et al. [MANDAL05] proposed a workflow scheduling algorithm in grids by ranking the discovered nodes according to their expected performances for a specific application component. Gounaris et al. [GOUNARIS04, GOUNARIS06b] proposed a resource scheduling method for parallel query processing in computational grids in which the partitioned parallelism problem is mainly examined. In [BOSE07], Bose et al. examined the problem of efficient resource allocation for a given set of parallel query sub-plans by taking advantage of the bushy tree representation of queries. Liu and Karimi

[LIU08] 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. 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. Silva et al. [SILVA06] 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 and realize resource allocation incrementally by monitoring the ongoing query execution performance. Venugopal et al. [VENUGOPAL06] proposed a grid service broker for scheduling e-science applications in data grids. They focused on adaptive scheduling algorithms and brokering strategies.

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 this class of RA algorithms, there can be found both static and dynamic resource allocation approaches. Nearly all the static resource allocation based methods [BOSE07, GOUNARIS04, GOUNARIS06b, KANT05, LIU08, MANDAL05] consider communication requirements of tasks by examining networking properties of resources; on the other hand, most of the dynamic resource allocation studies [SILVA06, VENUGOPAL06] 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.

2.3.2.2 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. According to our search on this class of methods, we realized that most of the existing studies are concerned with dynamic resource allocation [GOUNARIS05, GOUNARIS09, KOTOWSKI08, PATON09, RUIZ09], although there are few static resource allocation studies as well [SOE05]. Soe et al. [SOE05] 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 [GOUNARIS05], 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 [GOUNARIS04] and then allocation is refined according to the monitored query execution performance. Paton et al. [PATON09] 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. Kotowski et al. [KOTOWSKI08] 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* [MATTOSO05], which exploits query parallelism and load balancing. In [GOUNARIS09], 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. Ruiz et al. [RUIZ09] proposed a query allocation method for distributed information systems. Which is based on *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.

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.

2.3.2.3 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 2.5. 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 2.5 Qualitative Evaluation Summary

2.3.2.4 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, 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.

We have used the *GridSim* [SULISTIO08] 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, cost 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.

In Figure 2.3, 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 2.3. Allocation times for 5 and 20 tasks in a small scale environment varying from 10 to 60 nodes is shown in Figure 2.3.a

and Figure 2.3.b, respectively. In Figure 2.3.c and Figure 2.3.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 2.3, 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 2.4.

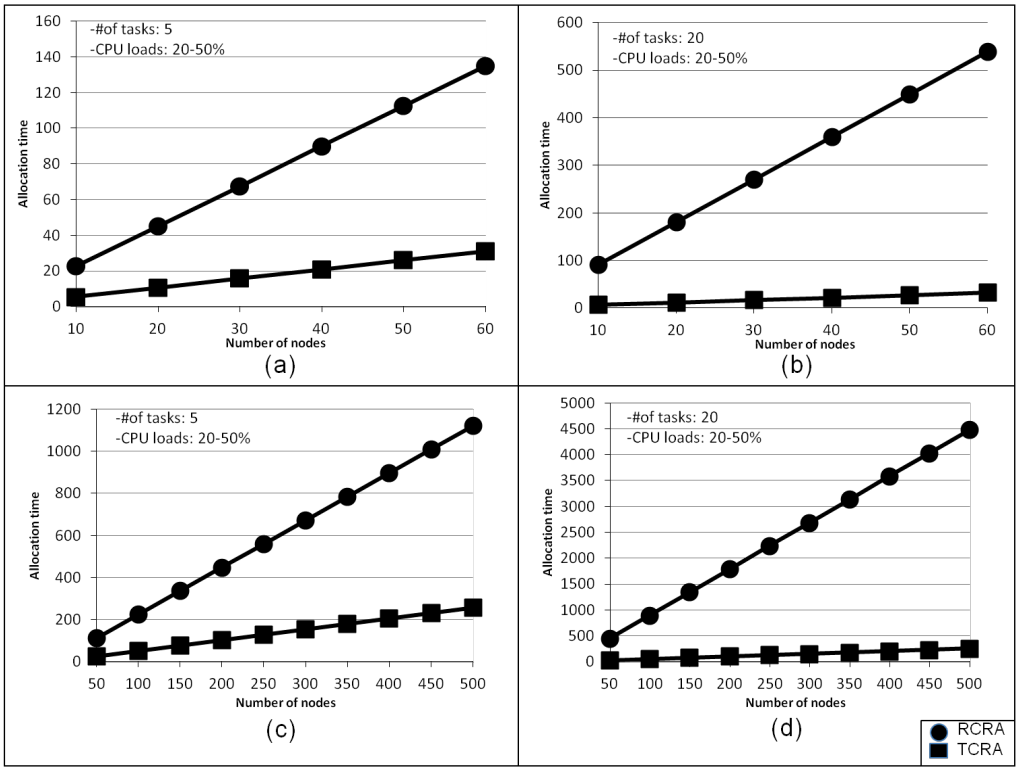

Figure 2.3 Allocation time for varying grid sizes.

It can be seen in Figure 2.4 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.

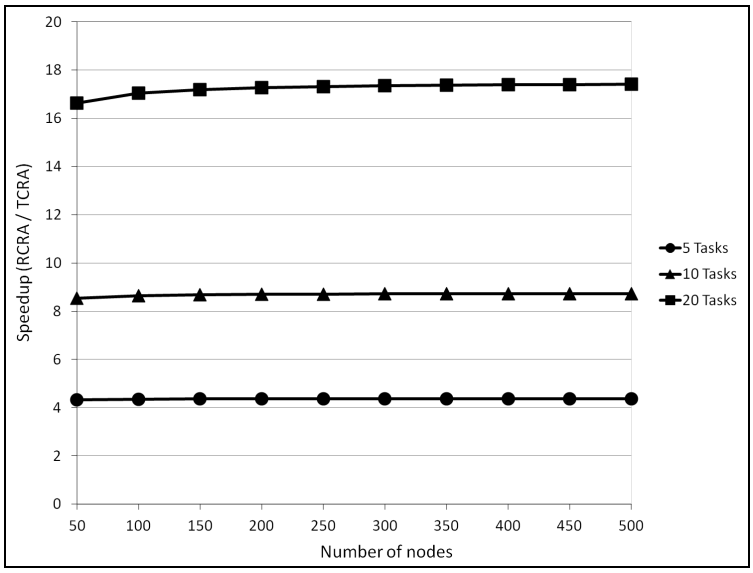

Figure 2.4 Speedup values for varying scale

In Figure 2.5, 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.

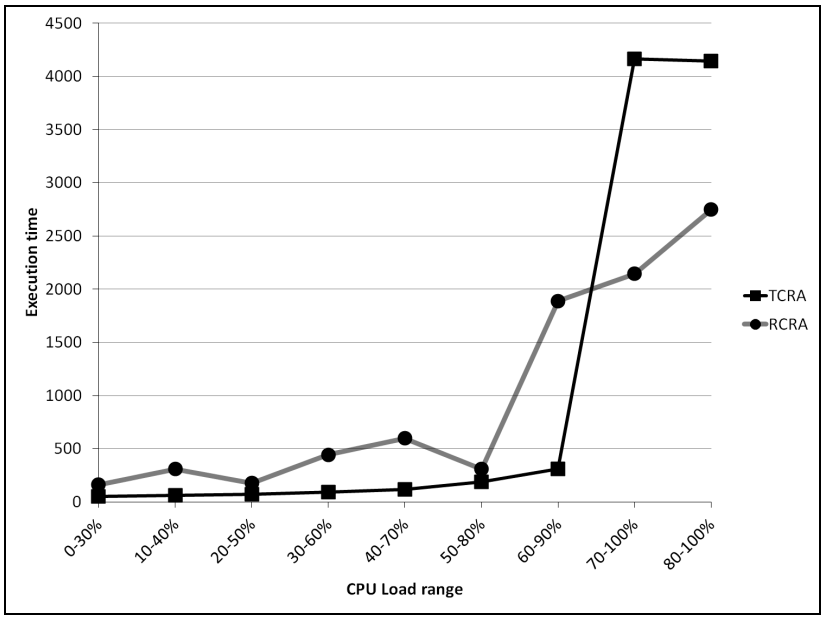

Figure 2.5 Execution time of tasks for varying CPU Loads

In Figure 2.6, 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 2.6, 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.

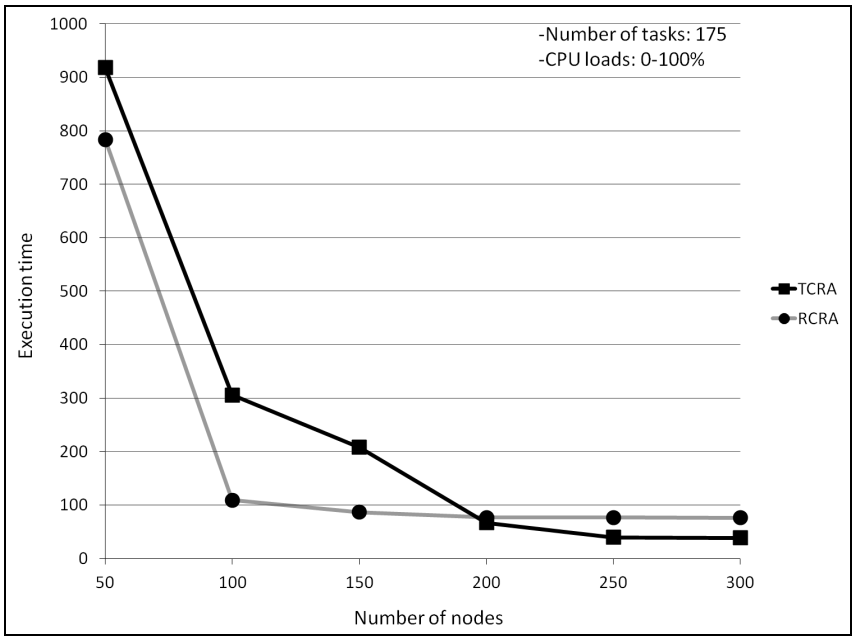

Figure 2.6 Execution time of tasks for varying number of nodes

2.3.3 Fault-tolerance in Query Processing in Grid Environments

Since grid environments are dynamic, eventual node failures are likely during the execution of queries. These failures may be very costly if the queries are long running and if the system is not designed fault-tolerant. Therefore fault-tolerance can be considered as a must in processing queries in grid environments. Since fault-tolerance involves in allocation of new nodes, we consider it as a part of the resource allocation algorithm.

In current literature, there can be found many resource allocation algorithms with dynamicity support [GOUNARIS05, SILVA06, VENUGOPAL06, KOTOWSKI08, GOUNARIS09, PATON09, RUIZ09]. Although these studies provide resource allocation methods in cases of dynamicity of nodes, none of them consider failure of nodes during execution of stateful operators in queries. Since stateful operators, such as hash join, require recovery of states of the nodes in case of node failures, dynamic resource allocation methods are not applicable in these cases. To the best of our knowledge, we find few studies for fault-tolerant query processing in grid environments [SMITH05, SMITH07, TAYLOR08, BESTEHORN10]. Although these studies provide fruitful algorithms, none of them is specialized on processing stateful query operators in grid environments. In this section, we present a brief literature survey related to resource allocation algorithms for fault-tolerant query processing in grid environments.

2.3.3.1 Recent Studies

There can be found many studies in the literature, which examine resource allocation for query processing in grid environments such as [GOUNARIS04, GOUNARIS05, KANT05, MANDAL05, SOE05, GOUNARIS06b, SILVA06, VENUGOPAL06, BOSE07, KOTOWSKI08, LIU08, GOUNARIS09, PATON09, RUIZ09]. These studies either present static resource allocation algorithms in which the resource allocation is performed once, and allocated nodes sustain execution until the tasks are completed; or they present dynamic resource allocation algorithms in which allocation is dynamically modified during the execution of the tasks according to the monitored status of the resources. Although dynamic resource allocation algorithms take dynamicity of the environment into account, none of these algorithms consider the case in which node failures occur. Still, there can be found many other studies in the current literature, which examine fault-tolerant distributed query processing in different environments [ABADI05, BALAZINSKA08, BESTEHORN10, CHANDRASEKARAN03, HWANG05, HWANG07, KWON08, SMITH05, SMITH07]. However, few of them are applicable in grid environments

especially for processing stateful query operators such as hash joins [BESTEHORN10].
Smith and Watson presented a fault-tolerant query processing system for distributed query processing in [SMITH05]. Their study is an extension to their previous work *OGSA-DQP* [ALPDEMIR04] with the addition of *fault detector* and *fault handler* modules. In their study, the queried node generates a query plan, performs an initial resource allocation and initiates the execution of the query. For the fault tolerance, the algorithm performs a rollback recovery protocol. In [SMITH07], Smith and Watson discuss failure recovery alternatives in query processing in grid environments. In their study, they examined three failure recovery alternatives namely *restart*, *reduce* and *replace*. Bestehorn et al. presented a fault-tolerant query processing algorithm in structured P2P systems in [BESTEHORN10]. In their study, they examined the query operations in two classes namely stateless and stateful operations. The proposed method examines fault-tolerance over these operations with two different perspectives: *i)* fault-tolerant routing and *ii)* replication. The study exploits the functionalities proposed by the CAN peer to peer system for selecting the backup peers. A very detailed survey study related to fault-tolerant distributed query processing can be found in [TAYLOR08]. In his study, Taylor examines different fault-tolerant distributed query processing algorithms in three classes namely: *i)* upstream backup [SMITH07, SMITH05], *ii)* active standby [ABADI05, BALAZINSKA08, CHANDRASEKARAN03] and *iii)* passive standby [HWANG05, HWANG07, KWON08].

2.3.3.2 Conclusions

Considering our analyses related to fault-tolerance in query processing, we have found many studies examining this problem. Although these studies present fruitful algorithms, to the best of our knowledge, we cannot find studies designed especially for grid environments, which examine fault-tolerance in stateful query operators. Most of the examined studies are focused on stream processing which does not include stateful operators. Moreover most of these studies are designed for P2P systems. Although there

are many similarities between grid systems and P2P systems, in many aspects they have slight differences. Therefore we believe that the characteristics of grid environments and requirements of query processing tasks should be focused in order to find suitable RA algorithms for fault-tolerant query processing in grid environments.

2.3.4 Conclusions for Resource Allocation Algorithms

In this section, we classified, analyzed, evaluated and compared different kinds of resource allocation methods in grid environments. 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. We also present a brief literature survey about resource allocation algorithms for fault-tolerant query processing in grid environments. Even though we have examined many fruitful methods in this section, we cannot find studies that provide a complete solution regarding both the grid environments' characteristics and query processing requirements.

The studies, which are examined in this section, 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 [BIZARRO09, IOANNIDIS97] in order to contribute to the query optimization by choosing the appropriate allocation method for the specific situation.

2.4 Overall Conclusions

In this chapter, we have analyzed, evaluated and compared the recent studies for resource discovery, self-stabilizing spanning tree construction, resource allocation and fault-tolerant query processing problems in grid environments by classification. With respect to our analyses, we believe that there are still open issues and potential

contributions for each of these problems. More precisely, regarding grid systems' characteristics, following open issues in each studied problem attract our attention:

i. Resource discovery for query processing in grid environments: Since grid system is dynamic, and since properties of resources change in time, in order to obtain up-to-date resource information, the RD algorithm should visit each node individually. In this manner, unstructured P2P systems based RD algorithms are considered to be very suitable for our purposes. However, since grid is a large-scale environment, the RD algorithm should be scalable and should avoid using flooding of messages. Therefore, we believe that without centralizing resources' information, a topology control mechanism can be used for efficient distribution of RD messages. On top of the generated topology, the resource discovery can be realized by the use of efficient message distribution such as broadcasting for request messages and converge-casting for response messages. Therefore, new resource discovery methods can be proposed, aiming at *i)* decreasing message and time complexities, *ii)* obtaining up-to-date resource information and *iii)* reliable resource discovery that is prone to node failures and topology changes. In this manner, self-stabilizing spanning tree structures might be used as a topology control mechanism with the addition of an efficient routing algorithm.

ii. Self-stabilizing spanning tree construction: The examined recent self-stabilizing spanning tree algorithms present fruitful solutions to the spanning tree construction problem. However, to the best of our knowledge, we have found few studies which aim at decreasing the diameter of the constructed spanning tree. This property aims at efficient distribution of messages through the whole grid environment. Therefore, new heuristical algorithms can be proposed for constructing spanning trees with smaller diameters.

iii. Resource allocation for query processing in grid environments: The resource allocation algorithms for query processing is roughly studied in the current literature. However, since grid is a relatively new concept, in general, existing studies are extensions to the previous resource allocation algorithms which are designed for distributed and parallel systems. To the best of our knowledge, we have found few studies which are

designed especially for resource allocation for query processing in grid systems. Still, we have found many missing contributions in the current literature such as allocating of nodes considering proximities to the data sources. We believe that, new algorithms can be designed for this purpose considering all characteristics of grid systems. Regarding our analyses on the current literature, we also find that the resource allocation algorithm should be designed in a bottom-up fashion, starting from allocation of resources for the most atomic tasks in queries. Therefore we believe that an initial resource allocation algorithm which is designed for only one query operator should be studied at first glance. Then, extensions to the designated algorithm can be designed in order to cover the whole query.

iv. Fault-tolerance for query processing in grid systems: The grid environment is characterized by its dynamic nature. However, the dynamicity characteristic of the grid should be examined in its two different meanings: *i)* dynamicity of properties of nodes and *ii)* dynamicity of nodes in terms of their existance in the environment. Regarding our literature survey, we found that most of the dynamic resource allocation algorithms examined the dynamicity characteristic of grid systems in its first meaning (*i*). However, its second meaning may have a big impact in the performance and reliability of query execution in cases where nodes fail frequently. We have found very few studies in the current literature which examined dynamicity of grid systems in its second meaning (*ii*). Therefore, we believe that valuable contributions can be made to the research community by examining fault-tolerance in this domain.

CHAPTER 3: RESOURCE DISCOVERY FOR QUERY PROCESSING IN GRID ENVIRONMENTS

Résumé

A travers ce chapitre, nous proposons un algorithme de découverte de ressources complet qui se compose de deux couches : i) Le contrôle de topologie pour la découverte de ressources dans les environnements Grille et ii) algorithme de la découverte de ressources basé des arbres couvrants. Premièrement nous proposons trois algorithmes de contrôle de topologie qui construisent des arbres couvrants auto-stabilisants, et qui peuvent être utilisés dans le découvert de ressources pour le traitement des requêtes en environnements Grille. Ensuite, nous proposons un algorithme de découverte de ressources en exploitant les arbres couvrants lors de la distribution des messages de découverte des ressources.

Abstract

In this chapter, we propose a complete resource discovery algorithm that is composed of two layers: i) topology control for resource discovery in grid environments and ii) spanning tree based resource discovery algorithm. We first propose three topology control algorithms that construct self-stabilizing spanning trees which can be used in the resource discovery for query processing in grid environments. Then, we propose a resource discovery algorithm by exploiting the spanning-tree during the distribution of resource discovery messages.

3.1 Introduction

Resource discovery problem in grid systems can be defined as searching and locating resource candidates that are suitable for a job, in spite of the dynamicity and large scale of the environment. 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 source that provides metadata and its contents such as database; or it may be a service which is programmed to accomplish a specific task. Effective usage of these resources in a grid system relies on the discovery of the right resources for given tasks. The main characteristics of the grid environment such as dynamicity, heterogeneity and large scale make the resource discovery a time consuming process, which can negatively affect the performance of query execution. The recent studies on grid resource discovery for query processing were deeply analyzed in Chapter 2. Regarding our analyses and conclusions, all different resource discovery approaches have their own advantages and disadvantages. In this chapter, we aim at designing a resource discovery method which brings some advantages of the existing approaches by extending them to overcome their drawbacks.

To summarize the existing studies, we have examined the current literature in grid resource discovery for query processing in three classes namely *i)* centralized/hierarchical systems based grid resource discovery, *ii)* P2P systems based grid resource discovery and *ii)* agent based grid resource discovery. Although centralized/hierarchical grid resource discovery methods become very popular in the first decade of grid environments, their centralized nature does not suit to dynamicity and large-scale characteristics of the grid environment. On the other hand, P2P systems based RD methods are heavily used in the current literature and they provide reliable RD algorithms and a decentralized control for the scalability. The most recent P2P techniques use structured P2P systems, which increase the performance of the resource discovery process drastically. Moreover, usage of the distributed hash table (DHT) mappings exploits the scalability and reliability properties of the P2P systems since all nodes in the system involve the resource discovery process. However, DHTs have some disadvantages for the resource discovery domain. The usage of DHTs limits the RD algorithms in terms of support for dynamic-attribute, multi-attribute and range queries. To overcome these problems, many algorithms proposed to use the topological structure of the overlay to efficiently distribute the RD queries directly between the resources. For these purposes, super peer systems mention the topological structuring

of resources. However, they use super peers mostly to act as central indexing servers, which brings them the disadvantage of being single point of failures.

Regarding aforementioned analyses, in this chapter, we first propose algorithms to structure the grid environment using efficient topological structures such as spanning trees in order to address scalability problems. We design the proposed algorithm in self-stabilizing fashion in order to address dynamicity characteristic of the grid system. Then, by exploiting the topological structure, we propose a resource discovery method, which broadcasts the RD queries over the spanning tree and which converge-casts the results through the queried node efficiently aiming at decreasing the complexity of the RD process.

The contribution of this chapter is twofold:

i. First, we propose three new self-stabilizing spanning tree construction algorithms aiming at decreasing the diameter of the resulting spanning tree. Then we compare our proposed algorithms to each other and to similar existing studies and choose the best performer algorithm for RD purposes.

ii. Second, we propose a new resource discovery algorithm, which broadcasts the RD queries to the whole environment and collects results by converge-casting the response messages by exploiting the generated spanning tree.

The rest of this chapter is organized as follows, in section 3.2, we propose three self-stabilizing spanning tree construction algorithms. We explain the algorithms in detail and analyze them by considering their time and message complexities. We evaluate our algorithms and provide quantitative comparisons between them and two existing similar algorithms by simulation. Then we discuss the results and show that the proposed algorithms perform better in many situations. In section 3.3, we propose the spanning tree based resource discovery algorithm for query processing in grid environments. We present the algorithm in detail, provide the complexity analyses and discuss the experimental evaluations by comparison. Lastly, in section 3.4, we present our conclusions.

3.2 Topology Control for Resource Discovery in Grid Environments

Spanning tree algorithms are widely used as topological control mechanisms in many distributed applications. A spanning tree is a subset *S* of a graph *G* which contains every node in *G* and which does not contain any cycles. With the growth in the scale of distributed systems such as grid systems, the need for such topological control mechanisms has gained significant importance. These control mechanisms decrease the complexity of distributed algorithms caused by the connectivity of the underlying graph. By using spanning trees, many distributed applications, especially those involving multicast or broadcast operations, can be optimized by making use of their properties.

Distributed spanning tree construction algorithms become very popular in the last decade since it is very hard to keep system wide information in large environments in order to build such topologies centrally. Many algorithms have been developed to build different types of spanning trees. Several studies focus on constructing minimum spanning trees in which the sum of edge weights is minimized [AHUJA89, AWERBUCH87, GALLAGHER83, LIEN88]. Minimum spanning trees are useful especially if communication costs are required to be minimized. Studies such as [BLIN09] construct minimum degree spanning trees in which degrees of vertices are minimized. This property helps efficient routing of messages under heavy communication traffic. Some other studies aim to construct minimum diameter spanning tree in which diameter of the resulting spanning tree is minimized [BUTELLE95]. This property helps broadcasting of messages by optimizing the distances between vertices in the graph.

Besides classical distributed spanning tree construction algorithms, some of these studies also consider dynamicity and fault tolerance in their design. Self-stabilizing spanning tree algorithms have gained importance recently [GARTNER03] by the wide spread usage of unstable systems such as grids. Self-stabilizing concept guarantees the validity of spanning tree structure without having the need to regenerate the spanning tree every time the structure has changed. Self-stabilization is a paradigm for distributed systems that allows

the system to achieve a desired global state, even in the presence of faults. The concept of self stabilization was introduced in 1974 by Dijkstra [DIJKSTRA74]. The idea of self-stabilizing algorithms is that independently of the global state of the system, the system will reach to a correct global state after a finite amount of time. In a self-stabilizing algorithm, each node maintains local variables, and changes its state according to only on its local variables and the contents of its neighbors' local variables. Each process checks these local variables continuously and takes corrective actions in case of failures. The contents of a node's local variables constitute its local state and the union of all local states constitutes the system wide global state. The self-stabilization property allows the system to stabilize by local corrections instead of system wide re-construction in case of failures. Thence, self-stabilization is a very useful approach for the systems in which dynamicity occurs frequently, such as grid systems. Therefore it may be convenient to use self-stabilization paradigm in such environments while designing distributed applications.

In this section, we propose new algorithms for self-stabilizing spanning tree construction that consider degrees of nodes in determining the root node. This heuristic is based on the observation that a root node will be involved in more frequent communication than the rest therefore it would be sensible that this node should have the property of having a higher degree than the average. We also aim to compare our algorithms with existing studies. For this purpose we selected two existing spanning tree construction algorithms which do not consider any complex spanning tree property, a classical [KSHEMKALYANI08] and a self-stabilizing spanning tree construction algorithm [AFEK91]. We have examined, analyzed, implemented and tested these algorithms, and compared the test results in terms of runtime of the algorithms and resulting spanning tree diameters.

The rest of this section is organized as follows: Section 3.2.1 examines the two selected existing spanning tree construction algorithms by giving detailed analysis. The three new self-stabilizing spanning tree algorithms are proposed in section 3.2.2. The correctness and complexity analyses of the proposed algorithms are provided in section 3.2.3. The implementation details of examined algorithms are given in section 3.2.4. Finally, in

section 3.2.5, conclusions, tradeoffs, comparisons, advantages and drawbacks of the proposed algorithms are discussed.

3.2.1 Selected Algorithms for Comparison

In this section, we examine and analyze two existing spanning tree construction algorithms in detail: memory-efficient self-stabilizing spanning tree algorithm (MEST) [AFEK91] and asynchronous concurrent initiator spanning tree algorithm (CIST) [KSHEMKALYANI08]. These algorithms will be used as reference to compare with our algorithms. The main difference between these two algorithms is the self-stabilization property.

3.2.1.1 Memory-efficient self-stabilizing spanning tree algorithm (MEST)

In [AFEK91], authors propose a self-stabilizing spanning tree construction algorithm with the assumption that nodes have unique identifiers and every node knows its neighbors. They also assume that nodes are aware of their neighbors' states; in other words, when a node fails, neighbors of the failed node notice this failure and update their neighbor lists. In this model, the authors do not consider any special node such as the root node initially. Every node runs the same algorithm. At the beginning, each node tries to construct a spanning tree rooted at itself. Then each node examines its neighbors' roots and selects the neighbor which has the biggest root as its parent. At the end, the biggest id node becomes the root of the final spanning tree.

Each node i has local variables indicating its neighborhood (N_i), its parent node (P_i), its root node (R_i) and its distance to the root node (D_i). In the global legal state, each node has the same root with the biggest node id in the graph, parents of nodes are within their neighborhood and distance of each node is 1 bigger than its parent's distance ($D_i = D_{parent} + 1$). The root node has distance 0, and points to itself as its root and its parent. The algorithm runs in an infinite loop and checks the legal state conditions continuously. To achieve self stabilization, each node compares its neighbors' roots with its own root. If any

neighbor has a bigger id root, then the node sends a *Request* message to the biggest id root through its neighbor in order to join its tree. When a root node receives such a message it replies with a *Grant* message and allows that node to join its tree.

The algorithm checks two conditions to determine whether it is in a legal state or not:

A: $[(R_i = i) \wedge (P_i = i) \wedge (D_i = 0)] \vee [(R_i > i) \wedge (P_i \in N_i) \wedge (R_i = R_{parent}) \wedge (Distance = D_{parent} + 1 > 0)]$

B: $A \wedge (R_i >= max(R_{Ni}))$

Condition A states that either the node is a root node; or the node is not a root node, its root is bigger than its id and same as its parent's root, its parent is in its neighborhood and its distance to the root is 1 bigger than its parent's distance. Condition B states that condition A holds true and node's root is the biggest among its neighbors' roots. When both A and B are true, the node is considered to be in the legal state. The algorithm checks these two conditions and takes action according to their correctness. When both conditions are false, then the node is in an illegal state and sets itself as the root node of its own tree. If condition A is true and node's root is not the biggest id node within its neighbors' roots, then the node chooses to join to the tree with bigger root id. To realize this, the node sends a *Request* message to the biggest id root within its neighbors' roots through its neighbor. If the condition B becomes true, then the node is in the legal state. The algorithm also takes other actions in order to relay messages while checking these conditions. In case of failure of an inner node in the tree, the children of the failed node notice this failure and switch to the illegal state because of the failure of condition A. This failure triggers self-stabilization property of this algorithm and relevant nodes take action in order to stabilize the spanning tree.

3.2.1.2 Asynchronous Concurrent Initiator Spanning Tree Algorithm (CIST)

The asynchronous concurrent initiator spanning tree algorithm [KSHEMKALYANI08] is a basic classical distributed spanning tree construction algorithm in which nodes only need to know their neighborhood information. The algorithm does not ensure self-

stabilization property. Each node starts to build its own tree rooted at itself at the beginning. When a node wants to initiate the algorithm as a root, then it sends a *Query* message to its neighborhood indicating that it is a root node. When a node receives a *Query* message it compares the id of the sender with the id of its root, if new root has a bigger id then the node changes its root to the new root. In this case if the node is a leaf node then it sends an *Accept* message, if not it sends a *Query* message to its neighborhood indicating its new root. If new root id is smaller than current root of the node, then the node sends a *Reject* message to the sender of the *Query* message. When a node receives an *Accept* message, then it adds the sender node to its children list. If that node is an initiator then it finishes its execution upon receiving *Accept* messages from all its neighbors, if the node is a relay node, it sends *Accept* message to its parent upon receiving *Accept* messages from all of its neighbors. When a node receives a *Reject* message, it sends an *Accept* message to its parent. Details of the algorithm can be found in [KSHEMKALYANI08].

3.2.2 Proposed Algorithms

In this section we propose three different self-stabilizing spanning tree algorithms. All our algorithms construct the spanning tree by considering the highest degree node when selecting the root of the tree. Nodes periodically send their variables to their neighbors in order to inform them about their status. We assume the existence of trusted communication channels between nodes in our algorithms. Only one hop information is required in our model.

All three algorithms check the same conditions during self-stabilization. The conditions are described below:

C1: The node is a legal root node (*myParent = Id, myRoot = myTag, myDistance = 0*)

C2: The node is a legal ordinary tree node (*myRoot > myTag, myParent is my neighbor, root of my parent = myRoot, myDistance = distance of my parent + 1*)

C3: The node is the one which has the biggest root among its closed neighborhood

3.2.2.1 Maximum Degree Self-Stabilizing Spanning Tree Algorithm 1 (MDST1)

We propose an extended version of MEST algorithm [AFEK91] by considering degrees of nodes while selecting the root node [COKUSLU10]. The MEST algorithm constructs the spanning tree according to the id of the nodes by choosing the biggest id node as the root node. This heuristic may have some disadvantages in complex networks because it does not consider the suitability of the chosen node as a root node. In many situations, predefined constraints are advantageous in choosing the root node. For instance, choosing the highest degree node as the root, or choosing the center of the graph, as the root node may be preferable if the desired operation is broadcasting or multicasting. On the other hand, simple and straightforward nature of MEST algorithm makes it preferable because of its low complexity measures. For these reasons, we first propose to modify the MEST algorithm so that it constructs a spanning tree rooted at the highest degree node. The assumption here is that the highest degree node in the graph is a better candidate to be a root node than a randomly chosen root. Because, this choice may decrease the diameter of the resulting spanning tree since the height of the tree will be decreased. To realize this, we first propose to use a simple hash function that combines degrees of nodes with their node ids to generate a new unique node identifier which is sorted by the degree of nodes. We called the resulting hash number as the *Tag* of a node. This function produces unique *Tag* values for each node by inserting the degree value to the most significant part and id values to the least significant part of the *Tag* number. It normalizes the least significant part by inserting zeros at the beginning of the node ids. The algorithm uses *Tag* values in determining the root node. The basic idea of our algorithm is similar to the MEST algorithm; the difference is the usage of *Tag* values instead of ids of the nodes in determining root node. At the end of the execution of the MDST1 algorithm, a spanning tree rooted at the highest degree node is constructed. The finite state machine of the algorithm can be seen in Figure 3.1. Each node starts its execution by setting its variables so that it becomes a root node. Then the node checks the three conditions and takes an action according to the results. If *C1* or *C2* is true and *C3* is false, it means that the node

has a neighbor with a bigger root. In this case the node chooses the maximum rooted neighbor and sends a *Request* message to that root via its neighbor and changes its state to *WAIT_GRANT*. The node changes its state to *TREE_NODE* upon receiving the *Grant* message. If a node, which is in *TREE_NODE* state discovers that both *C1* and *C2* are false then the node sets its variables so that it becomes a root and changes its state to *ROOT*. At the end of the algorithm, only the root node remains in *ROOT* state, and all other nodes change their states to *TREE_NODE* state. If any node fails, the neighbors of the failing node update their variables, and take action if rules of the tree structure are broken.

The main contribution of this algorithm is the usage of the maximum degree node as the root node and it aims at obtaining smaller diameter spanning trees, which may result in significant profit in large-scale environments.

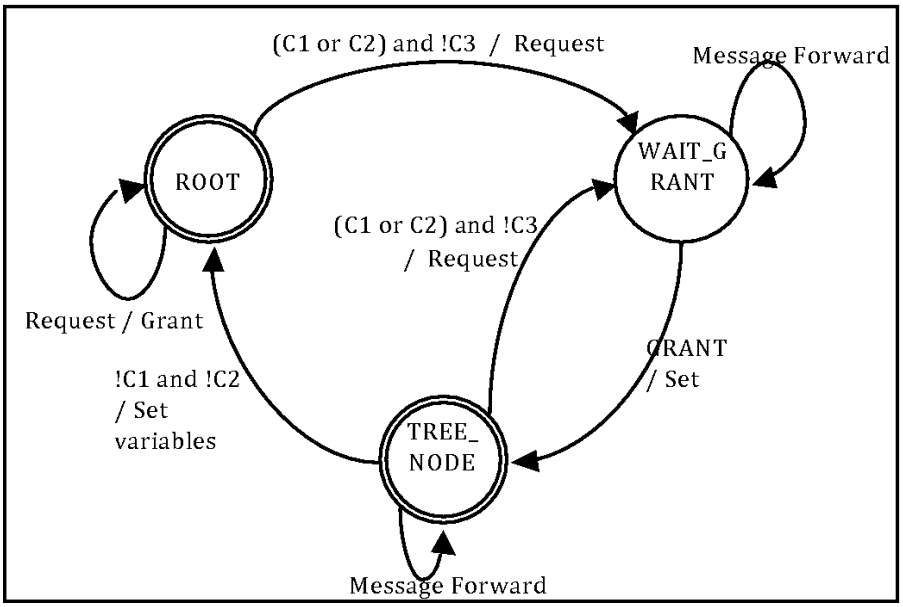

Figure 3.1 Finite State Machine of MDST1 Algorithm

3.2.2.2 Maximum Degree Self-Stabilizing Spanning Tree Algorithm 2 (MDST2)

In MDST1 algorithm, root selection and tree construction is held by a hand-shaking like protocol. The node that wants to set its root to a new node sends a *Request* message to that node and waits for a *Grant* message. In MDST2 we propose a completely new algorithm

aiming at decreasing communication between nodes. For this purpose we remove *Request* and *Grant* messages in MDST1. Instead of using these messages, nodes can freely set their variables without asking permission from any other nodes. The three conditions, *C1*, *C2* and *C3* are still in use in this algorithm, but additionally we introduce a fourth condition in order to increase efficiency of the algorithm. In MDST1 algorithm, even if there exists a simple failure in the nodes such as distance value errors, those nodes reset their variables and start the self-stabilizing algorithm from the beginning by declaring themselves as root nodes. This kind of error can be frequently caused by addition or removal of nodes in the spanning tree. These changes can result in alternations in the distance values in some child nodes without changing their roots and parents. In MDST2, we take a simple corrective action in case of that kind of failure by defining the condition 4:

- C4: The node is an ordinary tree node, but its distance value is false *(myRoot > myTag, myParent is my neighbor, root of my parent = myRoot, myDistance ≠ distance of my parent + 1)*

Each node starts its execution by setting its variables so that it becomes a root node. Then each node scans its neighbors' roots. If its root is not the highest *Tag* among its closed neighborhood *(C1* or *C2* is true and *C3* is false*)*, it sets its variables so that its root becomes the highest *Tag* root within its neighborhood. If the node finds that *C1* and *C2* conditions are false, it checks the condition *C4*. If *C4* is true, it means that a simple failure exists; in this case, the node sets its distance to one greater than its parent's distance. If *C4* is false, then the node declares itself as a root node and resets its variables. The algorithm can be seen in Algorithm 3.1.

Algorithm 3.1. MDST2 Algorithm

Input: *List of neighbors*
Output: Spanning tree constructed

```
1:  if (C1 OR C2) AND !C3 then
2:          Find maximum rooted neighbor
3:          if id of maximum root ≠ id of my root then
4:             myParent ← maximum rooted neighbor
5:                myRoot ← maximum root
6:                myDistance ← distance of my parent + 1
7:                myType ← TREE_NODE
8:          end if
9:  else if !C1 AND !C2 then
10:   if C4 then
11:      myDistance ← distance of my parent + 1
12:   else
13:      myRoot ← myTag
14:      myParent ← myId
15:      myDistance ← 0
16:      myType ← ROOT
17:   end if
18: end if
19: end
```

In this algorithm, we aimed at decreasing message complexity of MDST1 algorithm. We also noticed a shortcoming of both MEST and MDST1 algorithms and added a new condition in order to increase the efficacy. Regarding these modifications, the main contributions of the MDST2 algorithm can be stated as decreased message complexity and increased effectiveness. These contributions inspired us to propose a completely different algorithm than the existing MDST algorithm.

3.2.2.3 Maximum Degree Center Based Self-Stabilizing Spanning Tree Algorithm (MDCST)

In the MDCST algorithm we further aim to improve the MDST2 algorithm by considering the center of the tree. We assume that selecting center of the tree as the root node may improve the spanning tree by decreasing the communication hops in broadcasting messages. For that reason we add a self-stabilizing center finding algorithm to the MDST2. In order to assure self-stabilization assumptions, we run the MDST2 and center finding algorithms in sequence. In each sequence the center finding algorithm

changes the *Tag* values of the nodes according to the distance values of nodes from the leaves. The *Tag* values are generated by combination of three parameters in MDCST algorithm. Those parameters are *Height*, *Degree* and *Id* values respectively where the *Height* value for the node *v* indicates the height of the subtree rooted at the node *v*. For the center finding phase, we used the algorithm proposed by Karaata et al. [KARAATA94]. This algorithm starts by assigning 0 to the *Height* values of the leaf nodes. Non-leaf nodes select the second highest *Height* values within their neighborhood and set their *Heights* as 1 greater than that value. More details about the center finding algorithm can be found in [KARAATA94].

The MDCST algorithm runs the MDST2 algorithm first, and constructs a spanning tree rooted at the maximum degree node since all nodes have the same *Height* values initially. After the spanning tree is generated the center finding algorithm runs in order to calculate *Height* values of the nodes. After center finding algorithm completes its execution, *Tag* values are updated. At this stage, *Tag* values are sorted by *Height*, *Degree* and *Id* of the nodes respectively. Since MDST2 algorithm is self-stabilizing, and since *Tag* values are changed, the MDST2 takes action in order to maintain the spanning tree. This maintenance makes the tree rooted at the highest *Tag* value node which is the center of the initial tree.

The main contribution of this algorithm is the usage of the center node as the root. The center node is the closest node to any other nodes in the tree. A spanning tree, which is rooted at the center node, is expected to have smaller diameter and increased functionality in broadcast and multicast operations.

3.2.3 Analysis

In this section we provide correctness and complexity analyses for the three proposed algorithms, MDST1, MDST2 and MDCST. In the message complexity analyses, we do not consider periodical messages between nodes, which allow nodes to share their variables.

Theorem *The MDST1 algorithm constructs a correct spanning tree rooted at the biggest degree node in finite number of steps.*

Proof. MDST1 Algorithm is based on the MEST Algorithm which is proved in [AFEK91]. The MEST Algorithm is constructed on the assumption of existence of unique node *id's*. The only change in the MDST1 algorithm is the use of the *tag* values instead of node *id's*. The *tag* values are generated to be unique by considering the degrees and *id's* of nodes and are sorted by the degree of nodes. Since newly generated *tags* are unique, the proposed modification does not affect correctness of the algorithm.

Theorem *The MDST2 algorithm constructs a correct spanning tree rooted at the biggest degree node in finite steps*.

Proof. MDST2 algorithm is a variance of the MDST1 algorithm in which the messages are eliminated. Since the nodes share their variables with their neighbors periodically, any change in a node is propagated through the spanning tree over the nodes. Therefore, the periodic messaging between nodes replaces the messages in the MDST1 algorithm, which does not affect the correctness of the algorithm.

Theorem *The MDCST algorithm constructs a correct spanning tree rooted at the biggest degree node in finite steps beginning from the third phase*.

Proof. In MDCST algorithm the *tag* values of the nodes contain eccentricities, degrees and *id's* of nodes. Since the new *tags* are also constructed to be unique, the modifications do not conflict with any assumption of the previous algorithm. On the other hand, the first phase of the algorithm constructs an arbitrary spanning tree rooted at the biggest degree node since eccentricities are set to *0* initially. In the second phase, eccentricities are calculated and all components of the *tag* values are determined. From this stage, the MDCST algorithm executes in the same manner as the MDST2 algorithm using the *tag* values. Therefore the MDCST algorithm stabilizes after the third phase.

Theorem *MDST1 and MDST2 algorithms stabilize in h rounds where h is the height of the resulting spanning tree*.

Proof. In MDST1 and MDST2 algorithms, initially each node marks itself as the root of its own tree. In the first round, the neighbors of the root node stabilize. In the second round, 2-hop neighbors of the root are stabilized and so forth. Since the generated spanning

tree is a BFS tree, the most distant nodes stabilize latest. Since the most distant nodes are *h* hops away from the root, where *h* is the height of the generated spanning tree, the algorithm stabilizes in *h* rounds.

Theorem *The MDCST algorithm stabilizes in 3h rounds where h is the height of the spanning tree.*

Proof. In the first phase of the MDCST algorithm, MDST2 algorithm is executed to construct the initial spanning tree. In the second phase the center finding algorithm is executed in order to find the center of the initial tree. Lastly in the third phase, MDST2 algorithm is re-executed in order to finalize the construction of the spanning tree rooted at the center of the initial tree. Each of these phases requires *h* rounds to stabilize, therefore the MDCST algorithm stabilizes in *3h* rounds.

Theorem *The message complexity of the MDST1 algorithm is Θ(2*N*h) where N is the number of nodes and h is the height of the spanning tree.*

Proof. In each round of the MDST1 algorithm, each node sets its root and its parent by the exchange of two messages namely *request* and *grant*. Since there are *N* nodes and since the algorithm stabilizes in *h* rounds, the message complexity of the algorithm is *Θ(2*N*h)*.

Theorem *The message complexity of the MDST2 algorithm is Θ(N*h) where h is the height of the resulting spanning tree and N is the number of node in the grid system.*

Proof. The MDST2 algorithm stabilizes in *h* rounds. In each round a node sends 1 periodical update message to its neighbors. Since there are *N* nodes in the system, *N*h* messages are required to stabilize the algorithm.

Theorem *The message complexity of the MDCST algorithm is Θ(N*3h) where h is the height of the resulting spanning tree and N is the number of node in the grid system.*

Proof. The MDCST algorithm stabilizes in 3*h* rounds. In each round a node sends 1 periodical update message to its neighbors. Since there are *N* nodes in the system, *N*3h* messages are required to stabilize the algorithm.

3.2.4 Simulations

In this section, we have implemented and tested two existing (CIST, MEST) and three newly proposed algorithms (MDST1, MDST2, MDCST) for the spanning tree construction. We compare these algorithms regarding their runtime performances and resulting spanning tree diameters. We also implement a brute-force algorithm (BFA) to find the minimum diameter spanning trees in our scenarios in order to compare results of the proposed algorithms against minimum diameters. For the BFA, we first find all possible shortest path spanning trees in the graph. We then choose the minimum diameter of all these shortest path spanning trees to obtain diameter of the minimum diameter spanning tree, since the minimum diameter should be on one of the shortest path spanning trees.

We have implemented the algorithms in the network simulator ns2 [FALL10]. For the self-stabilizing algorithms, we have simulated failure detection module by using periodical messaging between the neighboring nodes. Each node periodically sends a message to all its neighbors indicating its variables (root, parent and distance). If any message is not received from a neighbor in 2 periods then the node assumes that this neighbor has failed, and takes action. In all algorithms, we used UDP protocol for messaging.

We have generated 8 experiment scenarios by using uniformly distributed random wired network topologies ranging from 100 to 800 nodes. In self-stabilizing algorithms, the periods of status updates and self-stabilizing loops affect the runtime of the algorithm drastically. For that reason, we tried to choose the update interval as small as possible that *ns2* allows. The same update interval is used in all experimentations to ensure fair comparison between algorithms.

Runtime results of the algorithms can be seen in Figure 3.2. The runtime of the MEST Algorithm [AFEK91] increases as the size of the network grows. This increase is due to the *Request* and *Grant* messages, which increase with the growth in the network. The CIST algorithm's runtime remains nearly constant as the network grows. This is because the algorithm is not self-stabilizing, and the neighborhood information is known priori.

MDST1 algorithm performs better than the MEST algorithm since MDST1 algorithm does not use messages in case of distance failures. Moreover addition of the degree heuristic may cause this improvement since a bigger degree node has more communication channels than other nodes that decrease the traffic load on single transmission channels. The MDST2 algorithm performs much better than all algorithms that are examined in this section. The prevention of usage of *Request* and *Grant* messages decreases the runtime of the algorithm drastically. On the other hand the MDCST algorithm performs somewhere between MDST2 and MDST1 algorithms. This is because of sequential execution of center finding and MDST2 algorithms. Figure 3.3 shows the diameters of the resulting spanning trees by using 5 different algorithms and the diameter of the minimum diameter spanning tree. The diameter of a tree is an important measure that is defined as the minimum distance between the two most distant nodes in the tree. It affects the performance of algorithms like routing and broadcasting algorithms, which use resulting spanning tree's properties.

It can be observed that both MEST [AFEK91] and CIST [KSHEMKALYANI08] algorithms have resulted in similar results in terms of tree diameters, while the others have resulted in smaller diameter spanning trees. This difference is caused by the degree heuristic that is used in choosing the root node. We observe similar tree diameters in both MDST1 and MDST2 algorithms since they use the same heuristic. The extra rule in MDST2 algorithm results in smaller diameter tree, which is very close to the diameter of the minimum diameter spanning tree. We also observe that both MDST1 and MDST2 algorithms choose the same nodes as root nodes, but the extra rule in MDST2 results in different child-parent relations in the resulting tree which provides a smaller diameter spanning tree. We expected that MDCST algorithm would generate better results than MDST1 and MDST2 algorithms since it uses the center of the tree as the root node. However, we observed that the results are not very different from MDST2 algorithm. This is because in many situations the highest degree node was located around the center of the tree.

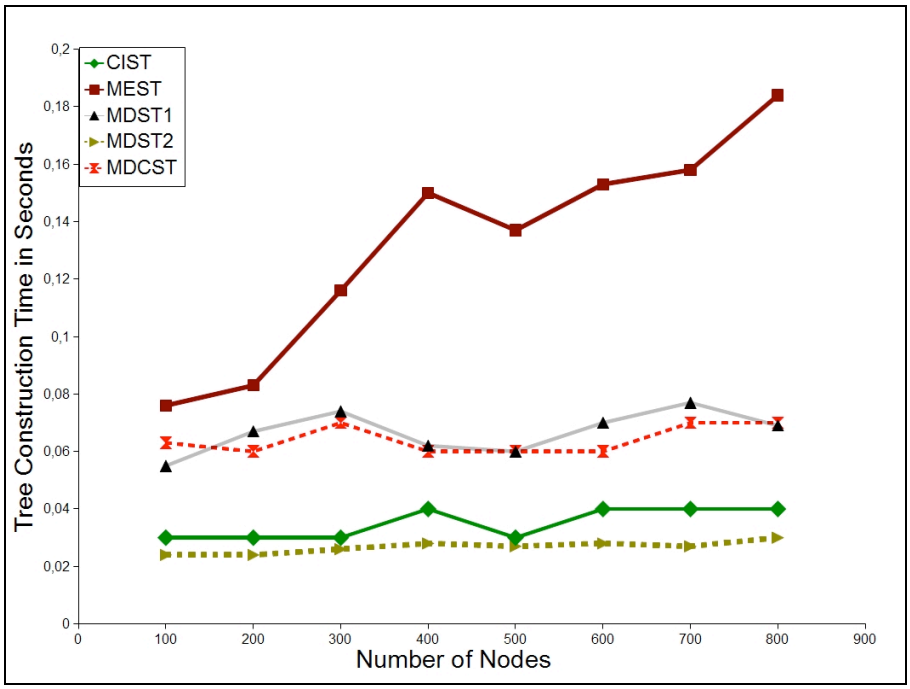

Figure 3.2 Runtime results of spanning tree construction algorithms

According to runtime and diameter results and comparisons we can say that MDST2 algorithm outperforms the remaining algorithms that are examined in this section. We also observed that addition of the center heuristic does not improve the test results as we expected.

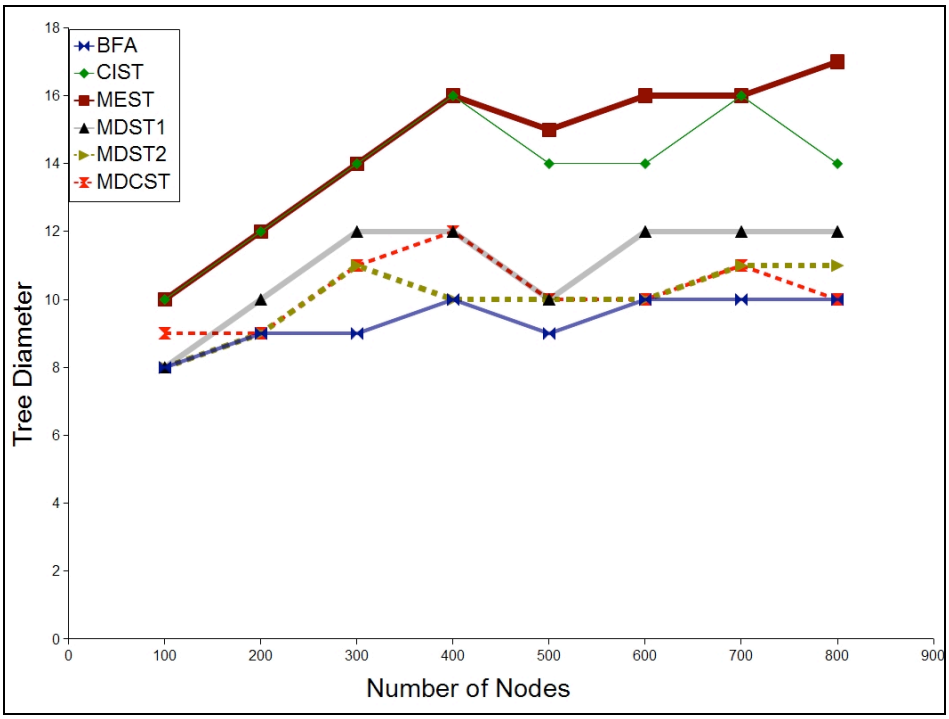

Figure 3.3 Diameter results of spanning tree construction algorithms

3.2.5 Conclusion

In this section, we proposed three new self-stabilizing spanning tree algorithms, which use the heuristic of choosing the highest degree node as the root. We have designed, implemented and tested our algorithms using network simulator ns2. In order to compare our algorithms with the existing algorithms, we also implemented and tested two existing spanning tree construction algorithms that rely on two different paradigms. We have compared simulation results of five algorithms in total. We showed the differences and similarities between the approaches in terms of runtime and tree diameter results. According to the implementation results, we can say that the two existing algorithms behave similarly in terms of the resulting spanning tree's degrees when the constraints are also similar. However, in MDST1 and MDST2 algorithms, with the use of maximum degree heuristic, the diameter of the resulting spanning tree is decreased. In MCDST algorithm, we realized that in complex network topologies, the root of the spanning tree,

which is generated by using degree heuristic, is located around the center of the tree in most scenarios. Therefore, we concluded that using the center of the tree as root node does not improve the resulting spanning tree significantly. According to the runtime results, we can say that our algorithms outperformed the other two compared existing algorithms.

According to our experimental observations, we can conclude that in dynamic environments such as grid systems, self-stabilizing algorithms can be used effectively. Moreover choosing the highest degree node as the root of the tree can improve the resulting spanning tree by decreasing the tree diameter.

3.3 Spanning Tree Based Resource Discovery for Query Processing in Grid Environments

Resource Discovery is one of the key issues in successful Grid systems. New methodologies for Resource Discovery are constantly researched due to the dynamicity, heterogeneity and large-scale characteristics of grid environments. Resource discovery (RD) in Grids can be defined as searching and locating resource candidates that are suitable for a job in which processing environments' constraints are clearly specified. On the other hand, the problem is defined as realizing the resource discovery in a reasonable time, considering the characteristics of the environment. The problems that may arise because of the dynamicity and large-scale properties of the environment were addressed in the section 3.2 by the use of self-stabilizing spanning trees. However, in order to discover up-to-date resource information in such a topology, resource discovery algorithms are required, which realizes efficient distribution of request and response messages related to resources.

There can be found many studies in the current literature for resource discovery in grid environments. Although these studies present fruitful solutions to the RD problem, to the best of our knowledge, we cannot find a complete solution, which addresses all requirements of query processing caused by the characteristics of grid systems. Requirements related to the scalability and dynamicity were addressed in previous section.

However, in order to obtain reliable up-to-date RD results, an RD algorithm should visit all nodes in the grid environment. We cannot find a resource discovery algorithm, which uses efficient topology control structures that are prone to dynamicity of the underlying environments. For these reasons, in this section, we propose the Spanning Tree Based Resource Discovery (STRD) algorithm, which exploits the self-stabilizing spanning tree that is generated by the MDST2 algorithm.

3.3.1 The Spanning Tree Based Resource Discovery (STRD) Algorithm

In STRD algorithm, we aim at efficiently discover resources by exploiting the underlying spanning-tree structure. In order to obtain reliable up-to-date results, the algorithm contacts with all nodes in the grid environment. The main contribution of the STRD algorithm is to efficiently scatter the resource discovery requirement lists and gather the results to the queried node exploiting the self-stabilizing spanning tree structure. For the STRD algorithm, we assume that the self-stabilizing spanning tree exists. In this spanning tree the nodes communicate with only their child and parent nodes.

Our algorithm starts its execution by the reception of a query by any arbitrary node in the grid environment. Such node analyses the query and prepares a requirement list (*RL*) for the resource discovery. The queried node sends the *RL* to the root of the spanning tree via its parent. When the root node receives the *RL*, it sends a message (*RD_Req*) to all its child nodes. This message is broadcasted to the spanning tree until it reaches to the leaf nodes. When a leaf node receives *RD_Req* message, it generates a response message (*RD*_Resp) and inserts its id into this message if it meets the requirements for the resource discovery. Then it sends the *RD_Resp* message to its parent. The parent node waits until all its child nodes respond to the *RD_Req* message. When all child nodes respond, it merges all responses, including its own response, to a new *RD_Resp* message, and sends this message to its parent. The *RD_Resp* messages are converge-casted upwards until they reach the root node. When the converge-cast is completed, the root node relays the result, which includes

all discovered resources, to the queried node. The flow of the algorithm is shown in Figure 3.4.

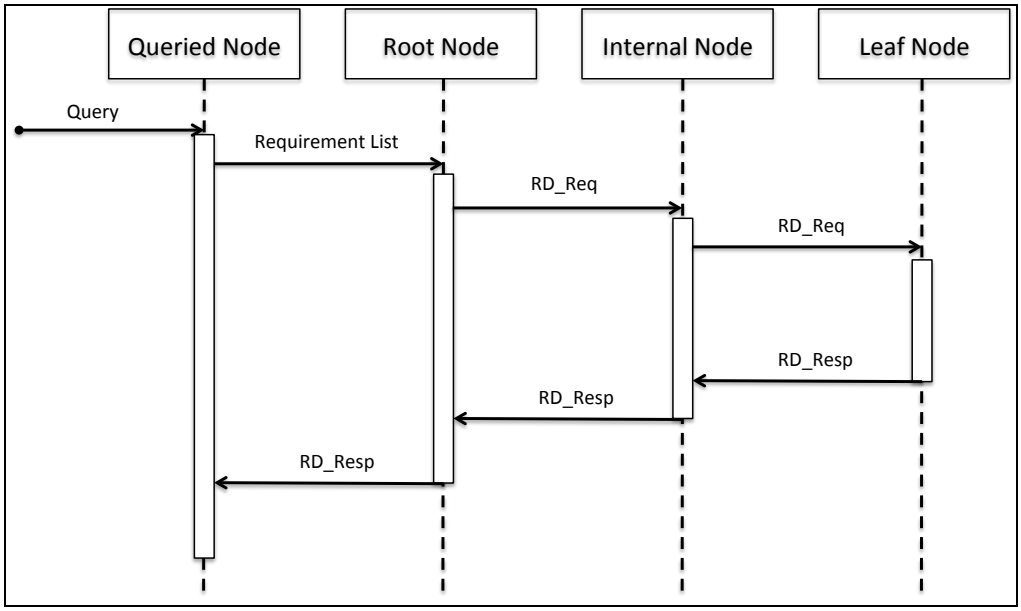

Figure 3.4 The STRD Algorithm Sequence Diagram

3.3.2 Analysis

The STDR algorithm exploits the underlying spanning tree. Therefore it shares many important properties with the underlying topology. In this section we provide reliability and complexity analyses for the STRD algorithm.

Theorem *The STRD algorithm returns up-to-date results for the resource discovery without having false negative errors*.

Proof. The STRD algorithm runs over the self-stabilizing spanning tree, which is constructed by one of the proposed self-stabilizing spanning tree construction algorithms in section 3.2. Since the constructed tree is a spanning tree, by definition, it contains all the nodes in the grid environment. All these nodes involve the STRD algorithm by responding to the resource discovery request by considering their properties. Therefore the algorithm does not contain false-negative errors. Since the resource discovery request is processed in

each node by itself, the resulting information is always considered to be up-to-date. Moreover, the underlying topology is self-stabilizing, which means that eventual node failures do not affect the correctness of the topology and the STRD algorithm in consequence.

Theorem *The message complexity of the STRD algorithm is $\Theta(2h+2N)$ where N is the number of nodes in the spanning tree and h is the height of the spanning tree.*

Proof. The STRD algorithm starts its execution by the reception of a query. Following this, the queried node sends a requirement list message to the root node through the spanning tree. This phase results h messages to be sent, where h is the height of the spanning tree. In the second phase, the root node distributes a resource discovery request message to all nodes in the spanning tree, which results in N messages to be sent to the network, where N is the number of nodes in the spanning tree. After all these messages are distributed, nodes converge-cast response messages. This phase results in N messages to be distributed since the response messages are merged in parent nodes. When the response message reaches to the root node, it is sent to the queried node through the spanning tree, which results in h messages to be sent. At the end, the whole process requires $2h+2N$ message exchanges.

Theorem *The time complexity of the STRD algorithm is $\Theta(4h)$ where h is the height of the spanning tree.*

Proof. Following the reception of query by an arbitrary node in the grid environment, the queried node sends a requirement list message to the root node through the spanning tree. This phase requires h rounds for the message to reach to the root node where h is the height of the spanning tree. In the second phase, the root node distributes a resource discovery request message to all nodes in the spanning tree, which requires h rounds since the messages are sent in parallel by the nodes, which are at the same level. After the messages are distributed, nodes converge-cast response messages which requires again h rounds similarly. At the last step, the root node relays the final response message to the

queried node, which is *h* hops away. In total, the STRD algorithm requires *4h* rounds for the completion.

3.3.3 Simulations

In this section, we have implemented and tested the STRD algorithm. We also implemented a flooding based resource discovery algorithm (FBRD) to be used as comparison purposes. In FBRD algorithm, similar to [IAMNITCHI03], the resource discovery request is flooded to the grid environment starting from the queried node. Then, each node that meets the requirements sends response messages to the queried node. We compare the two algorithms regarding their runtime performances to complete resource discovery.

We have implemented the algorithms in the network simulator ns2 [FALL10]. We have generated 7 experiment scenarios by using uniformly distributed random wired network topologies ranging from 100 to 700 nodes. In STRD algorithm the self-stabilizing spanning tree is constructed beforehand. We have randomly chosen 20 nodes as data sources in each scenario. We discovered these resources regarding existence of the required data. We also discovered many properties of these resources alongside the data existence.

In Figure 3.5, the time required to discover resources is measured. The query is posted to a randomly chosen node in the system. As it can be seen in the simulation results, the runtime of the STRD algorithm remains nearly constant in the testing scenario, whereas the runtime of FBRD algorithm increases drastically as the number of nodes increases. The time complexity of the STRD algorithm is bound by the height of the spanning tree. For that reason, although the number of nodes increases, the performance of the algorithm remains nearly constant as soon as the height of the spanning tree does not change. On the other hand, the time and message complexities of the FBRD algorithm is $O(N^2)$, therefore the increase in the number of nodes results in a quadratic increase in the runtime of the resource discovery algorithm.

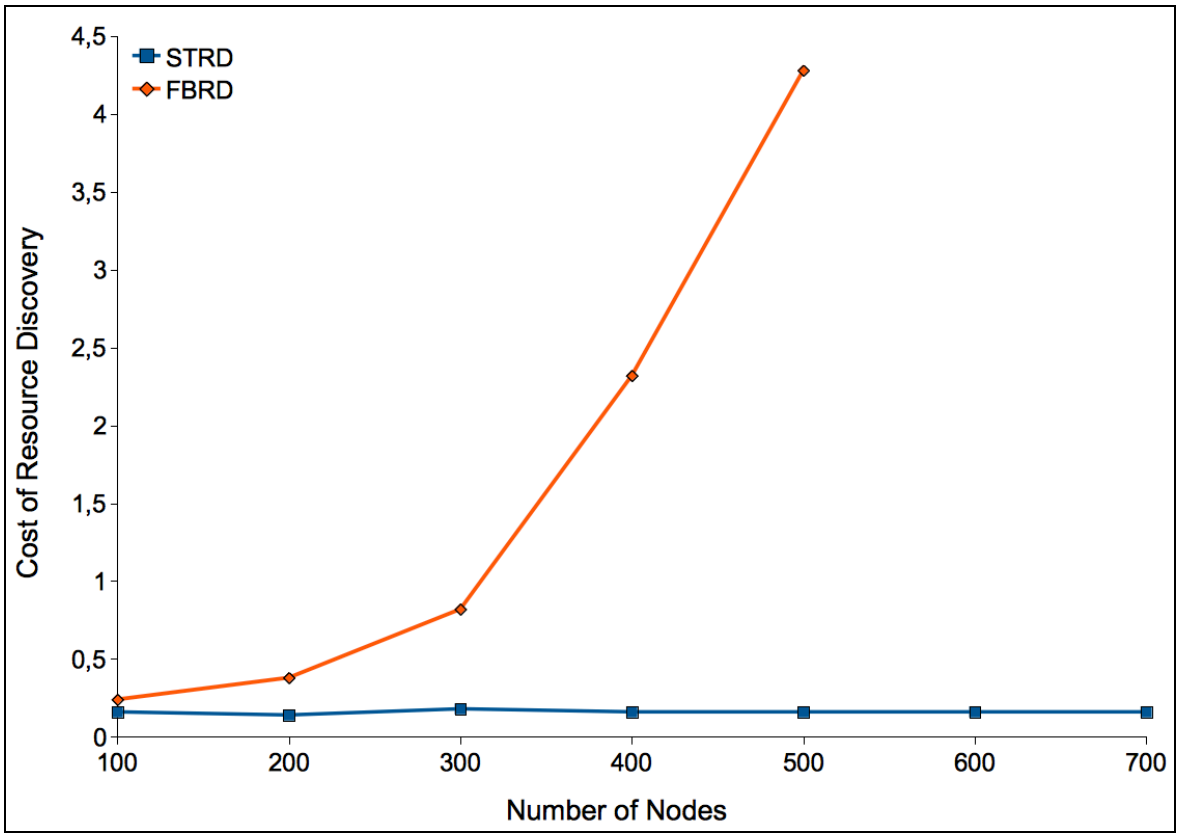

Figure 3.5 Runtime of the STRD algorithm

3.3.4 Conclusion

In this section we proposed the Spanning Tree Based Resource Discovery algorithm, which relies on the existence of a self-stabilizing spanning tree structure in the grid environment. In the STRD algorithm we exploited the self-stabilizing spanning tree construction algorithms that are proposed in the previous sections. We presented our algorithm in detail and provided the reliability and complexity analyses. We have consolidated our analyses with simulation and provided comparisons with an algorithm that uses flooding during the resource discovery.

Regarding our analyses and simulation results, we can conclude that the STRD algorithm meets nearly all requirements listed in the section 3.1. Since the algorithm uses self-

stabilizing spanning tree structure, it is prone to node failures. The efficient broadcasting and converge-casting of resource discovery messages increases the scalability of the algorithm. Moreover, since the algorithm does not use any limitations for the dissemination of resource discovery messages, it efficiently reaches to every node in the grid environment. Therefore it can be considered to be reliable in terms of correctness of the resource discovery results. Since the resource discovery messages are distributed to all nodes without having complex structures such as DHT's, the algorithm supports different types of queries such as multi-attribute, dynamic-attribute and range queries.

3.4 Overall Conclusions

Resource discovery constitutes a very important stage in query processing in grid environments. Differently from classical resource discovery algorithms, in grid environments and for query processing purposes, there can be listed many important requirements that should be met by a resource discovery algorithm. These requirements are listed and detailed in Chapter 2 as being complexity, scalability, dynamicity, reliability, support for multi-attribute, dynamic-attribute and range queries. Regarding the presented literature survey in Chapter 2, we concluded that the most reliable way to discover resources, in order to obtain up-to-date resource information, is to contact each resource individually in the grid environment. However, this method raises scalability problems since the grid system is large scale. Most of the existing studies solve the scalability problems by limiting the dispersion of resource discovery messages by using some metrics such as hop counts. However this method causes false negative errors since the resource discovery message cannot reach to every node in the system. Other methods use hierarchical structuring in the environment by keeping information of groups of resources in selected nodes. But this time the results become out-of-date since dynamic properties of resources are updated periodically in the selected nodes. Although there exist many studies in the current literature using different techniques, each have their own advantages and drawbacks. To the best of our knowledge, we find few algorithms that use topological

structures in distributing the resource discovery messages. For these reasons, in this chapter, we first proposed three self-stabilizing spanning tree construction algorithms. We provided correctness and complexity analyses of the proposed algorithms. We have tested our algorithms in ns2 simulation environment. We have also implemented two existing spanning tree construction algorithms for comparison purposes. We have compared in total 5 algorithms. We have discussed the weaknesses and strengths of our algorithms and showed the situations in which our algorithms outperform the similar existing studies. We then proposed a spanning tree based resource discovery algorithm that uses one of the proposed self-stabilizing spanning tree construction algorithms as a topology control mechanism. We provided reliability and complexity analyses for the proposed algorithm. We consolidated our analyses with simulation results. We have implemented and tested our algorithm and compared with a similar existing study.

We showed that the proposed resource discovery algorithm, in conjunction with the proposed self-stabilizing spanning tree construction algorithms, provide a complete solution regarding the requirements of query processing in grid environments. Our solution address the problems caused by the dynamicity of the environment by the use of self-stabilizing concept in the topology control layer. We address the scalability problems by the use of a spanning tree, which allows our algorithm to efficiently distribute the resource discovery messages. The proposed STRD algorithm aims at decreasing the complexity of the resource discovery process by using converge-casting of RD messages over the spanning tree. Lastly, we ensure support for dynamic attribute, multi-attribute and range queries by contacting each resource in the grid environment.

CHAPTER 4: RESOURCE ALLOCATION FOR QUERY PROCESSING IN GRID ENVIRONMENTS

Résumé

Nous proposons, dans ce quatrième chapitre, de nouveaux algorithmes d'allocation des ressources pour le traitement des requêtes dans les environnements Grille, compte tenu des proximités de nœuds candidats aux sources de données. Nous limitons d'abord, l'espace de recherche pour les ressources candidats. Ensuite, nous affectons des nœuds en commençant par le plus proche des nœuds aux sources de données. Nous présentons des analyses théoriques des algorithmes proposés et puis nous consoliderons ces analyses avec les simulations.

Abstract

In this chapter, we propose new resource allocation algorithms for query processing in grid environments considering proximities of candidate nodes to the data sources. We first limit the search space for the candidate resources. Then, we allocate nodes starting from the closest node to the data sources. We provide theoretical analyses of the proposed algorithms and consolidate analyses with the simulations.

4.1 Introduction

The number of different domains that exploit the facilities of grid environments increases everyday as the grid systems become more popular. Their large computing and storage capabilities attract many researchers' attention and leads them to propose new methods to port their existing computing environments to the grid systems [FOSTER04b]. Distributed query processing is one of these domains in which there exists large amounts of ongoing research to port the underlying environment from distributed and parallel systems to the grid systems [HUANG03, ANTONIOLETTI05, PACITTI07,

KOTOWSKI08, TANIAR08]. However, grid system's characteristics reveal many problems. It is generally assumed that grid systems have very large number of resources. These resources may correspond to computational resources such as CPU, memory, storage unit or network; they may be data sources, which provide metadata and its contents such as database; or they may be services, which accomplish specific tasks. On the other hand, a node corresponds to a computer in the grid, which contains some of those resources with a set of characteristics. To efficiently execute queries in the grid environment, suitable allocation of resources is essential. Resource allocation (RA) can be defined as selecting and allocating suitable nodes for executing query operators aiming at optimizing the runtime of the query execution. The optimum allocation of resources for query processing in large scale environments is proved to be NP-Complete [WANG90]. For this reason, existing studies search near optimal solutions for the RA problem using heuristic approaches. Resource allocation determines how many of the candidate resources will be used, and which tasks will be executed by which resources. These issues may drastically affect the performance of the query execution in grid environments. There can be found many studies in the current literature which address the problem of resource allocation for query processing in grid systems [GOUNARIS04, GOUNARIS05, SOE05, GOUNARIS06b, SILVA06, BOSE07, KOTOWSKI08, LIU08, GOUNARIS09]. Several survey studies examine and evaluate these studies with classification [COSTA08, EPIMAKHOV11, COKUSLU12]. Although the existing resource allocation studies provide interesting solutions to this problem, to the best of our knowledge, none of these studies consider decreasing the scale of the search space for the candidate resources. Moreover, we have found few studies which focus on the communication costs during the resource allocation [BOSE07, LIU08]. Thus, we believe that there are still some open issues that are not mentioned completely regarding the grid systems' characteristics.

In this chapter, we aim at designing a resource allocation algorithm for query processing in grid environments. For this, we first propose the Single Join Operator Resource Allocation (SJORA) algorithm that realizes resource allocation for queries consisting of a single join

operator [COKUSLU12_b]. In SJORA, we first generate a reduced search space for the candidate nodes. Then, we create an initial allocation plan considering proximity of the candidate nodes to the data sources. After SJORA, we further extend our algorithm and propose Multi-Join Resource Allocation (MJORA) algorithm, which allocates resources for queries consisting of multiple join operators. In MJORA, assuming the existence of the tree representation of queries, we traverse the query tree and, exploiting the SJORA algorithm, realize the resource allocation for the queries consisting of multiple join operators.

The contribution of this study is twofold. First, we address scalability of the resource allocation method by decreasing the size of the search space for candidate resources. The second contribution addresses the heterogeneity problem by selecting candidate nodes according to their proximities to the data sources aiming at decreasing data transfer costs.

Throughout this study, we assume that the relations, which are involved in the join operator, are horizontally partitioned into the grid where each partition resides in only one node. The existing partitioning may or may not be based on the join attribute. Therefore repartitioning may be required during the execution of the join operator. We examine a join operator as it consists of two atomic tasks namely scan and join. Scan tasks act as providers to the join tasks by reading the data from the storage unit and sending this data to the corresponding join tasks. Considering data locality constraint, we assume that the scan tasks are executed on the nodes where the partitions of the base relations reside. We call such nodes as scan nodes. The scan nodes may either be nodes that hold base relations, or they may be nodes that are already allocated for another join operator that produces temporary relations to the current join operator. On the other hand, we call the nodes in which join tasks are executed as join nodes. Lastly, the node that receives the query at the beginning will be called as queried node for the rest of this chapter.

The rest of this chapter is structured as follows. In section 4.2, we introduce the Single Join Operator Resource Allocation algorithm in detail. In section 4.3, we present the Multi Join Resource Allocation Algorithm. Finally in section 4.4, conclusions are presented.

4.2 Single Join Operator Resource Allocation (SJORA) Algorithm

In this section, we propose the Single Join Operator Resource Allocation (SJORA) algorithm for queries consisting of one join operator [COKUSLU12_b]. In the SJORA algorithm, we aim at finding suitable nodes for the scan and join tasks, which compose the join operator. Due to the data locality constraint, the scan tasks are allocated at the nodes in which the partitions of relations reside. However, the search space for the candidates of the join tasks is very large since there are no strict constraints beforehand for the join nodes. For this reason, we first aim at reducing the search space for the candidates of join nodes. To realize this, we designed the SJORA algorithm as consisting of two complementary algorithms. In this manner, we first propose Proximity Based Candidate List Generation (PBCG) Algorithm in section 4.2.1. After determining the list of candidate nodes, we propose Join Task Resource Allocation (JTRA) Algorithm in section 4.2.2, which determines the parallelization degree and finalizes the allocation of resources.

4.2.1 Proximity Based Candidate List Generation (PBCG) Algorithm

The resource allocation for join tasks is not a straightforward process since every node in the grid environment is practically a candidate resource for these tasks. In a large-scale environment, it might not be suitable to consider such a large number of nodes as candidates for the join task since this may cause performance degradation for the resource allocation process. Although there might be some constraints to decrease the number of candidates for join tasks, such as hardware or software constraints, we believe that the most profitable constraint would be the proximity of candidates to the scan nodes. For that reason, in PBCG algorithm, we try to refine the set of candidates by choosing the set of nodes that are closer to the scan nodes in the grid. To realize this, we start flooding a message beginning from each scan node. The first node that collects flooded messages from all scan nodes is considered to be located at the center of the scan nodes. The algorithm in the queried node as shown in Algorithm 4.1. At the beginning, the algorithm

sends a *startPBCG* message to all scan nodes, which causes them to start flooding operation (line 1). After sending this message, queried node waits for the candidate nodes to respond to the flooded messages. Upon receiving a message from a candidate, the queried node adds the sender to the candidate list (line 3). The first replied candidate is considered to be the closest candidate to all scan nodes. In line 4, the algorithm checks the termination condition. Since the queried node is not aware when the flooding ends, it has to use a termination condition to finalize the algorithm. This allows the queried node to decide when to stop waiting for new messages for the PBCG algorithm. In our algorithm, we used the most distant scan nodes (*nodeA* and *nodeB*) for determining the termination condition. Receiving a *PBCGCandidate* message from one of these two nodes simply means that the flooded messages are spread at least to all scan nodes. To realize this, if the replied candidate is one of the most distant scan nodes, queried node terminates the algorithm and finalizes the candidate list.

Algorithm 4.1. PBCG Algorithm in the queried node

Input: (*i*) The list of scan nodes
(*ii*) The most distant scan nodes (*nodeA* and *nodeB*)
Output: List of candidate nodes

1: *send startPBCG message to all scan nodes*
2: *upon receiving a PBCGCandidate message:*
3: *add sender to the PBCGCandidateList*
4: **if** *sender = nodeA or nodeB* **then**
5: *terminate*
6: **else**
7: *continue receiving messages*
8: **endif**
9: **end**

The algorithm, which is run in other nodes, is shown in Algorithm 4.2. When a node in the grid system receives a message, it checks its type. If the message type is *startPBCG*, the node starts the flooding by sending a *PBCGFlooding* message to all its neighbors (line 3). The termination condition for flooding is embedded in this message as *hopLimit* value, which is the hop count between the most distant scan nodes. If the received message type

is *PBCGFlooding,* the node extracts *hopCount* value from the message and increments by one (line 6). It then adds origin of the message to the *receivedScanNodes* list (line 7). If the *receivedScanNodes* list contains all scan nodes, the node sends a *PBCGCandidate* message to the queried node, which indicates that it is a candidate for the join task (line 9). The node then checks if the message should be relayed (line 11); if the termination condition is not met yet, the node relays the message by sending it to all its neighbors except the sender of the message (line 12). Each time the message is relayed, it is diffused to other nodes in the grid. The flooding operation continues *hopLimit* hops away from the scan nodes. This ensures that at least *t* nodes will be candidate for the join operation where *t* is the number of scan nodes.

Algorithm 4.2. PBCG Algorithm in other nodes

```
1: Upon receiving a message:
2: if message type is startPBCG then
3:     send PBCGFlooding message to all neighbors
4: else if message type is PBCGFlooding
5:     extract hopCount and hopLimit values from the message
6:     increment hopCount by one
7:     add origin of the message into the receivedScanNodes
8:     if receivedScanNodes includes all scan nodes then
9:         send PBCGCandidate message to the queried node
10:    end if
11:    if hopCount < hopLimit then
12:        relay PBCGFlooding message to all neighbors except the sender
of the      message
13:    else
14:        stop flooding the message
15:    end if
16: end if
17: end
```

4.2.2 Join Task Resource Allocation (JTRA) Algorithm

The basic execution of queries with partitioned parallelism can be realized by the allocation of a single node for the join task initially. On the other hand, parallelization of the join task might drastically increase or decrease the performance of the query execution

depending on the characteristics of the allocated nodes. For this reason, determining parallelization degree for a join operator is very important in query processing in grid environments, in which resources are heterogeneous. There are many different parameters that affect the performance of execution of a query in such environments. A resource allocation algorithm, which covers all these parameters, may cause excessive computation time to decide the parallelization degree and which nodes to allocate. Therefore, heuristically selected parameters are generally used in the current resource allocation algorithms [GAROFALAKIS97, GOMOLUCH03, SLIMANI04, MANDAL05, BOSE07, LIU08]. We believe that, in grid environments, data transmission costs are the determining factor in execution time of a query. Therefore, in JTRA algorithm, we propose a resource allocation algorithm, which considers the data transmission costs as the decision function for the parallelization degree. Although there are some similarities between the JTRA algorithm and the algorithm which is proposed in [GOUNARIS06b], the difference is that, our algorithm allocates nodes starting from the closest nodes to the scan nodes which ensures smaller data transfer costs. Another difference is that, our algorithm includes a decision function for the parallelism degree, based on the estimated data transfer costs, which struggles the heterogeneity issues in grid environment.

In JTRA algorithm, we use the candidate list, which is generated by the PBCG algorithm. This list contains candidate resources, which are closer to the scan nodes. The top of the list contains the closest candidate whereas the bottom parts contain more distant candidates. The JTRA algorithm can be seen in Algorithm 3. We start JTRA algorithm by taking a node from the top of the candidate list (line 2), which is located at the center of the scan nodes. Then we measure communication speeds between the selected node and the scan nodes (line 3). This measurement is realized on the go by the use of round-trip-time *(RTT)*. After gathering communication speed information, the algorithm calculates the new estimated data transfer cost. The addition of another candidate increases parallelization degree of the join task. As the parallelization degree increases, the amount of data to be transferred to each join node decreases. However, since newly selected candidates get

more distant, the lastly added node will have a poorer communication capability. This reveals a trade-off between decreased amount of data transfer and increased data transfer costs.

Algorithm 4.3. JTRA Algorithm

Input: (*i*) The list of join candidates (*candidateList*)
(ii) The metadata about the relations
Output: *List of nodes that are allocated for the join task*

```
1: do
2:    nodeC ← take a node from top of the candidateList
3:    measure connection speeds between all scan nodes and nodeC
4:   newEstimation ← estimate new data transmission time
5:    if newEstimation < queryRuntimeEstimation then
6:       add nodeC to the selectedNodes list
7:      queryRuntimeEstimation ← newEstimation
8:    end if
9: while candidateList is not empty
10: allocate selected nodes for the join task
11: end
```

Since the the size of the search space for the candidate nodes is limited by the use of the PBGC algorithm, the JTRA algorithm iterates the whole candidate list. In each iteration, the data transfer cost of the join operator is estimated. If addition of a candidate does not lead to a performance increase, the algorithm skips the addition of that candidate and iterates through the next candidate in the list. At the end of the algorithm, the best sorted resource combination within the candidate list is allocated for the join task.

The estimation of data transfer costs in JTRA algorithm consists of three parameters: *(i)* communication speeds between scan and join nodes *(S_{ij})*, *(ii)* local bandwidth of each join node *(B_j)*, *(iii)* sizes of partitions in each scan node (*$|P_i|$*). From those parameters, B_j and $|P_i|$ are provided by the resource discovery step. However, S_{ij} is determined on-the-go each time a new candidate node is added. Even if the measurements for S_{ij} are accurate individually, at the runtime, they might differ from the measured values when all scan nodes send their data to the join nodes concurrently. In such cases, the local bandwidth of a

join node might become insufficient to meet all incoming packets. In such cases, the congestion control mechanism of the underlying communication protocol regularizes the transfer rate of the sender nodes. In today's networking environments, most of the communication is handled by the TCP protocol, which provides its own congestion control mechanism. The main idea behind TCP's congestion control is to ask senders to decrease transmission rate for a specified amount if congestion occurs. This amount is generally determined proportionally regarding to the percentage of the sender's transmission over the entire traffic [FALL10]. Therefore, we heuristically normalize the S_{ij} measurements beforehand as appears in Algorithm 4.4.

Algorithm 4.4. Connection speed measurements normalization algorithm

Input: (*i*) *Measured connection speeds between scan and join nodes (S_{ij})*
(*ii*) *Local bandwidth of join node (B_j)*

Output: *Normalized list of S_{ij}*

1: $sum_j \leftarrow \sum_{i=0->n} S_{ij}$
2: $difference \leftarrow B_j - sum_j$
3: **if** $difference > 0$ **then**
4: **for** $i = 0$ *to* n **do**
5: $S_{ij} = S_{ij} - (difference * (S_{ij} / sum_j)$
6: **end for**
7: **end if**
8: **end**

Assuming that the partitions are evenly distributed on the scan nodes and redistribution of tuples will be uniform, the data transfer time between n scan nodes and m join nodes will be bounded by the slowest communication link. In such a setting, the data to be transferred from each scan node i to a join node will be $\{|P_i| / m\}$. Therefore the data transfer time estimation can be constructed as follows.

$$\max_{\substack{i=0->n \\ j=0->m}} \frac{|P_i|}{m * S_{ij}} \qquad (1)$$

The equation (1) returns the cost for data transmission between scan and join nodes considering the slowest parallel portion of the join task. Since all the remaining parallel portions of join task should wait for the slowest portion, it is the determining portion for the entire data transmission costs of the examined join task.

4.2.3 Analysis

In this section we provide time and message complexity analyses of the proposed algorithms, PBCG and JTRA.

Theorem 1. *The PBCG algorithm has O(d) time complexity, where d is the diameter of the network.*

Proof. PBCG algorithm uses the distance between the most distant scan nodes for the termination condition. In the worst-case scenario, the distance between the two most distant nodes in the network is the diameter of the network. Since the algorithm propagates *d* hops away from each scan node, the time complexity of the algorithm is *O(d)* where *d* is the diameter of the network.

Theorem 2. *The PBCG algorithm has $O(nN^2)$ message complexity, where n is the number of the scan nodes and N is the number of the nodes in the grid system.*

Proof. In PBCG algorithm, each scan node initiates a flooding operation originated from itself. Therefore there are *n* messages to be flooded to the network. In the worst case, the flooding operation lasts until all the nodes in the network receives flooded messages. Since each flooding operation has N^2 message complexity, the total worst case message complexity of the algorithm is $O(nN^2)$.

Theorem 3. *The worst case time complexity of the JTRA algorithm is O(nN), where n is the number of scan nodes and N is the number of nodes in the network.*

Proof. The JTRA algorithm uses the candidate list, which is generated by the PBCG algorithm. In the worst case, the list contains all nodes in the network. For each of these candidates, the JTRA algorithm measures the connection speed between the candidate and *n* scan nodes by sending them RTT messages. Therefore, the worst case time complexity of the algorithm is bounded by the number of messages which is *O(nN)*.

Theorem 4. *The worst case message complexity of the JTRA algorithm is O(nN), where n is the number of scan nodes and N is the number of nodes in the network.*

Proof. The JTRA algorithm uses two messages for measuring the communication speed between each candidate and scan nodes. Since in the worst case the algorithm uses *N* candidates, the total number of message exchange is *2nN* which can be expresses as *O(nN)* in the big o notation.

4.2.4 Simulations

In this section we present quantitative evaluation of the Single Join Operator Resource Allocation (SJORA) algorithm by comparing it with a comparative algorithm (CA) which reflects the common properties of the recent resource allocation algorithms such as [GOUNARIS04, GOUNARIS06b]. The main idea behind the CA is very similar to the algorithm proposed by Gounaris et al. [GOUNARIS04, GOUNARIS06b]. The algorithm ranks the nodes in the grid according to their properties. In our case, the most important property that influences the simulation results is the connection speed of the nodes. Therefore the CA algorithm ranks the nodes according to their connection speeds. Then the ranked nodes are sorted and the algorithm starts to allocate nodes starting from the top of the list. When addition of a new node does not lead to a performance increase, the algorithm terminates. We have implemented SJORA and CA algorithms in ns2 simulation environment and collected results for the cost of resource allocation process and duration of query execution.

We have generated grid simulation scenarios consisting of 100 through 700 nodes. Each node in the scenario represents a uni-processor computer in the grid system that has arbitrary connections to other nodes in the grid environment. The bandwidths of duplex connections between nodes are randomly assigned between 1 and 10 Gbps. We have randomly determined 20 scan nodes in each scenario. The distribution of these scan nodes are realized randomly over the simulated environment. Each scan node is assumed to store a partition of a base relation. The size of partitions in each node is assumed to be 50

GBytes and each scan node stores only one partition.

We have collected test results for the cost of resource allocation process and the time required to complete the query execution. Figure 4.1, shows the cost of the resource allocation process. As it can be seen in Figure 4.1, the cost of the SJORA algorithm is higher than the CA. This is because the SJORA algorithm processes its entire candidate list to find the best possible resource allocation within its candidates.

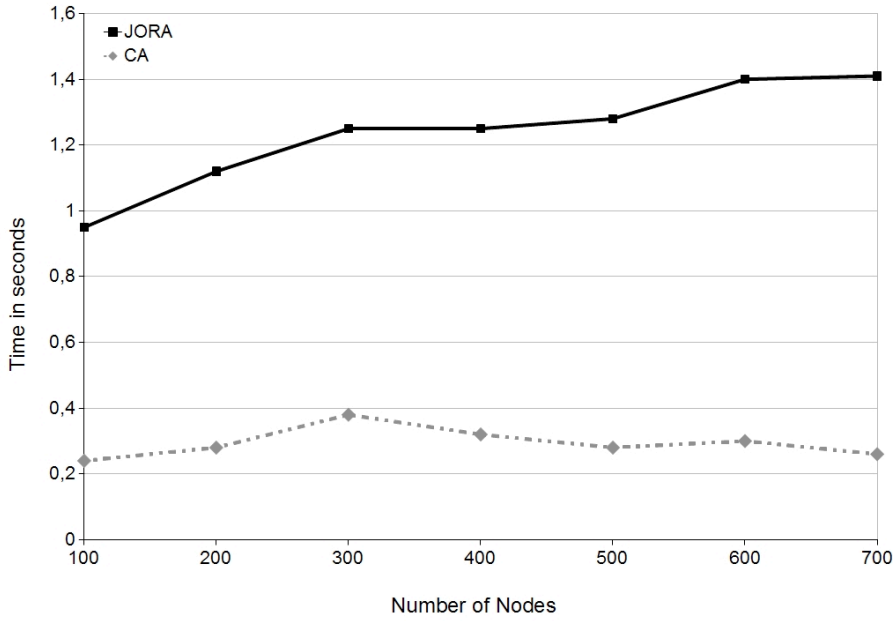

Figure 4.1 Cost for the resource allocation process

On the other hand, CA stops adding new resources whenever its performance increase drops below a certain threshold value. The approach used by CA may miss better resource allocation combinations with higher number of resources. However it is conceptually impossible to evaluate all possible resource allocation combinations.

Figure 4.2 shows the simulated query execution durations for the simulated query. In the figure, it can be seen that the resources allocated by the SJORA algorithm execute the query faster than the resources that are allocated by the CA. This is because, although the

resources that are allocated by the CA are the highest ranked nodes in the grid, they might be placed far from the scan nodes, which may result in slower data transfer rates. On the other hand, the resources that are allocated by the SJORA algorithm are closer to the scan nodes. For that reason, SJORA algorithm ensures allocation of more effective resources in terms of the communication performances with the scan nodes. This heuristic results in SJORA algorithm outperforming the CA.

Regarding the simulation results, which are shown in figures 4.1 and 4.2, the SJORA algorithm is more preferable if the resource allocation costs do not exceed the estimated query execution durations. In our simulation scenarios, the durations of query executions are much higher than the costs of the resource allocation processes. Therefore, in such cases, the SJORA algorithm might be considered as a better alternative to the existing resource allocation algorithms that are based on ranking functions.

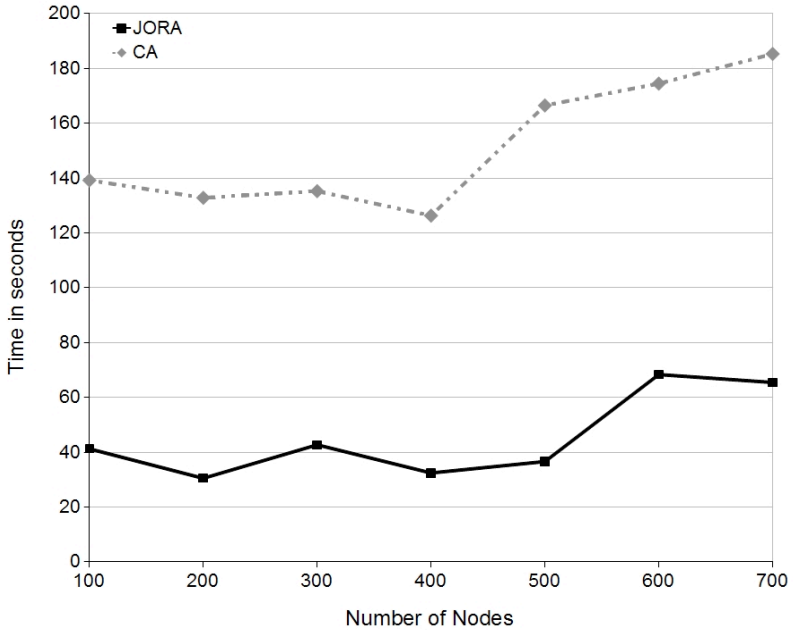

Figure 4.2 Time required to complete the query execution

4.2.5 Conclusion

In this section, we proposed a Single Join Operator Resource Allocation (SJORA) algorithm, which generates a finite candidate resource list by exploiting proximities of the candidates to the scan nodes. We presented our algorithm in detail and provided complexity analyses. Then, we strengthened our perspectives by the use of quantitative analyses and simulations. We showed that our algorithm outperforms the algorithms that use ranking functions without having the proximity information to the data sources.

Regarding the simulation results, we conclude that the SJORA algorithm might be a strong alternative to the existing resource allocation algorithms in many cases in which queries deal with large amount of distributed data. As indicated in its name, the SJORA algorithm is a resource allocation algorithm for queries consisting of a single join operator. The resource allocation algorithm for multiple join operator queries is presented in the next section.

4.3 Resource Allocation for a Multi Join Query

In this section, we extend the SJORA algorithm and propose the Multi Join Resource Allocation (MJORA) algorithm for queries consisting of multiple join operators. In the MJORA algorithm, we aim at finding suitable nodes for all tasks, which compose the join operators in the entire query. However, since the residing nodes of the temporary relations are not determined beforehand, we design the MJORA algorithm in bottom-up fashion starting from allocation of resources to the operators that use base relations. The rest of this section is organized as follows: In section 4.3.1, we propose the MJORA algorithm. In section 4.3.2, we present the analyses of the proposed algorithm. In section 4.3.3, we consolidate our analyses with the simulation results. Finally in section 4.3.4, we present our conclusions.

4.3.1 Multi Join Resource Allocation (MJORA) Algorithm

In this section, we examine hash join queries, which consist of one or more join

operators, as a use case for query operators. We assume that the optimized query operator tree is provided explicitly. The relations that are involved in the query are assumed to be horizontally partitioned into the grid without replication. We consider a query as consisting of hash join operators which are composed of atomic tasks namely scan, build and probe. Scan tasks act as providers to the build and probe tasks by reading tuples from their storage units and sending to corresponding tasks. Build tasks receive tuples from their scan tasks and build hash tables for that operator. Probe tasks are blocked during the execution of their corresponding build tasks. After build tasks complete, probe tasks start receiving tuples from their corresponding scan tasks and check for matching tuples in the hash table. Probe tasks pipeline matched tuples through their successor tasks in the query tree. More detailed explanation of the hash join operator in distributed environments can be found in [OZSU11].

In section 4.2, we find and allocate suitable resources for queries consisting of a single join operator assuming scan tasks are already allocated. Since queries may consist of more than one join operator, it is necessary to extend the SJORA algorithm by materializing the allocation of all tasks in the entire query. In this section, we propose the Multi Join Resource Allocation (MJORA) algorithm, which allocates nodes for the entire query that consists of multiple join operators. For the MJORA algorithm, we use tree representation of queries. An example query with its tree representation is shown in Figure 4.3. Vertices in the query tree represent tasks and directed edges represent data flow between tasks. In MJORA algorithm, we consider that queries consist of join operators, and join operators consist of scan, build and probe tasks. Each join operator in the query may be composed of more than one pair of build and probe tasks that execute concurrently over the different data partitions. The build and probe tasks in each pair are tightly coupled. Therefore we decided to allocate each pair of build and probe tasks in the same resource. For simplicity, in the rest of this paper, these pairs of build and probe tasks are named as join tasks.

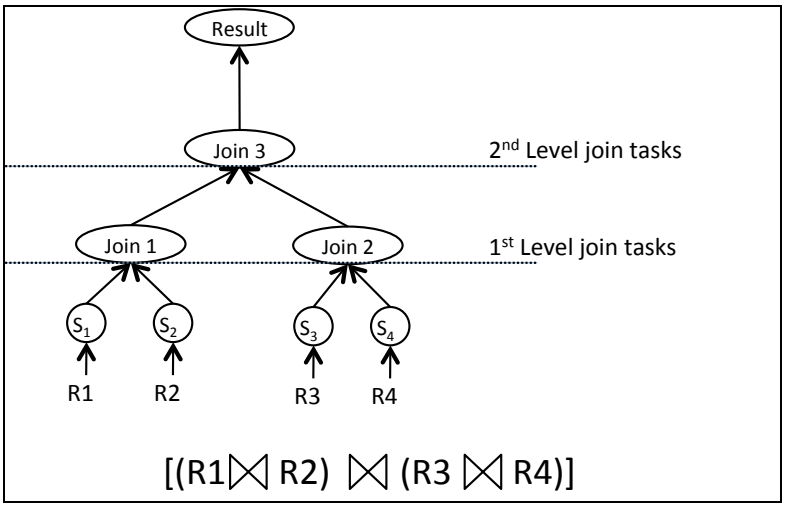

Figure 4.3 An example query and its query tree

Previously, in SJORA algorithm [COKUSLU12_b], we assumed that the partitions of base relations are known at the end of the resource discovery stage. However, for the temporary relations, it is not possible to find out their physical locations before precedent join tasks are allocated. Therefore, in MJORA algorithm, we execute the SJORA algorithm starting from the join tasks that are located at the lowest level of the query tree. These kinds of join tasks are marked as 1st level join tasks in Figure 4.3. Then, we allocate the join tasks that are located at the 2nd level in the query tree. We repeat this process until all tasks are allocated in the query. To realize this, we apply post-order tree traversal on the query tree. The generation of the query tree involves query optimization steps such as query reordering. Since this kind of optimization is out of scope of this thesis, we assume that the query tree is already provided exclusively. The MJORA algorithm is executed by the queried node. A snapshot of the initial resource allocation is stored in the query tree that resides on the queried node. Each vertex in the query tree contains a data structure called *treeElement*, which is shown in Figure 4.4. The *treeElement* contains four principal fields namely *type, list, left* and *right*. *Type* field indicates the type of the task whether it is a scan

or a join task. *List* field contains a list of allocated nodes for the task, which is initially empty. *Left* and *right* fields point to the left and right subtrees in the query tree.

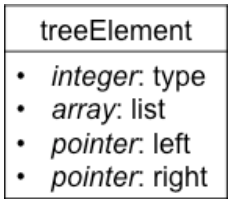

Figure 4.4 Data structure for the treeElement

The MJORA algorithm is shown in Algorithm 5. The algorithm runs recursively a post-order tree traversal (lines 1 to 6). In line 7, the algorithm processes the visited vertex. If the visited vertex is a scan task, MJORA algorithm fills list field of the *treeElement* with the list of nodes in which the partitions of the processed relation reside (line 8). Else, if the visited vertex is a join task, the list field is filled with the results of the SJORA algorithm. In this step, the list of partitions of base or temporary relations is provided by the child vertices (line 10). The estimated sizes of the temporary relations (*sizes*) are assumed to be provided exclusively. The most distant nodes and the distance between them for the partitions of temporary relations (*nodeA, nodeB and dist)* are provided by the topology control algorithm of the resource discovery module [COKUSLU10]. At the end of execution of the MJORA algorithm, all the vertices in the query tree contain the list of selected nodes for the corresponding task. At this stage, the queried node sends the query tree to all selected nodes in order to complete allocation of nodes and to start execution of the query.

Algorithm 4.5. MJORA algorithm

Input: *Root of the query tree*
Output: List of nodes to be allocated for each operator

```
1: MJORAAlgorithm(treeNode)
2:    if treeNode is empty then
3:       return
4:    else
5:      MJORAAlgorithm(treeNode.left)
6:      MJORAAlgorithm(treeNode.right)
7:      if treeNode.type = SCAN then
8:         treeNode.list = nodes in which partitions reside
9:      else if treeNode.type = JOIN then
10:          treeNode.list = SJORA(treeNode.left.list ∪ treeNode.right.list,
nodeA, nodeB, dist, sizes)
11:    end if
12:  end if
13:  return
14: end
```

4.3.2 Analysis

In this section we provide time and message complexity analyses of the proposed algorithm, MJORA.

Theorem 1. *The MJORA algorithm has O(jnN) time complexity, where j is the number of join operators in the query, n is the number of scan nodes and N is the number of nodes in the grid system.*

Proof. MJORA algorithm uses the SJORA algorithm for each join operator in the query tree. Since the time complexity of the SJORA algorithm is *O(nN),* the time complexity of the MJORA algorithm is *O(jnN)* for a query that consists of *j* join operators.

Theorem 2. *The MJORA algorithm has $O(jnN^2)$ message complexity, where j is the number of join operators in the query, n is the number of the scan nodes and N is the number of the nodes in the grid system.*

Proof. The MJORA algorithm uses the SJORA algorithm for each join operator in the query tree. Since the message complexity of the SJORA algorithm is $O(nN^2)$, the total message complexity of the MJORA algorithm is $O(jnN^2)$ for a query that consists *j* join operators.

4.3.3 Simulations

In this section we present evaluation of the Multi-Join Resource Allocation (MJORA) algorithm by simulation. We compare our algorithm with a comparative algorithm (CA) which reflects the common properties of the recent resource allocation algorithms such as [GOUNARIS04, GOUNARIS06b]. The main idea behind the CA is very similar to the algorithm proposed by Gounaris et al. [GOUNARIS04, GOUNARIS06b]. The algorithm ranks the nodes in the grid according to their properties. In our case, the most important property that influences the simulation results is the connection speed of the nodes. Therefore the CA algorithm ranks the nodes according to their connection speeds. Then the ranked nodes are sorted and the algorithm starts to allocate nodes starting from the top of the list. When addition of a new node does not lead to a performance increase, the algorithm terminates. The CA algorithm traverses the query tree and allocates resources for each join operator in the query. We have implemented MJORA and CA algorithms in ns2 simulation environment and measured the cost of resource allocation process and duration of execution of a sample query.

We have generated grid simulation scenarios consisting of 100 through 800 nodes. Each node in the scenario represents a uni-processor computer in the grid system that has arbitrary connections to other nodes in the grid environment. The bandwidths of duplex connections between nodes are randomly assigned between 1 and 10 Gbps. In our simulation scenario, we have simulated resource allocation and execution of a sample query consisting of 3 join operators, which joins 4 relations in total. The formal representation of the sample query is shown in the figure 4.5 where R1, R2, R3 and R4 present relations and $\bowtie$ presents join operator.

$$\{(R1 \bowtie R2) \bowtie (R3 \bowtie R4)\}$$

Figure 4.5 Formal representation of the sample query

Each relation is horizontally partitioned into 5 arbitrary scan nodes in our scenarios. The

distribution of the scan nodes is realized randomly over the simulated environment. Each scan node is assumed to store a partition of a base relation of size 50 GBytes. Each scan node stores only one partition.

We have collected test results for the cost of resource allocation process and duration of query execution. Figure 4.6, shows the cost of the resource allocation process for the entire query. As it can be seen in Figure 4.6, the cost of the MJORA algorithm is higher than the CA. This is because the MJORA algorithm processes its entire candidate list to find the best possible resource allocation within its candidates and measures the communication speeds from each candidate node to all scan nodes while calculating the estimated query duration. This overhead results in the MJORA performs slower than the CA in return for a better selection of resources. However, it can be seen in the figure that the MJORA algorithm scales well with the number of nodes in the grid environment. The cost of resource allocation process remains nearly constant as the number of nodes increase. This is caused by the limitation of the candidate resource search space. In each scenario the algorithm examines nearly the same number of candidate resource in the MJORA.

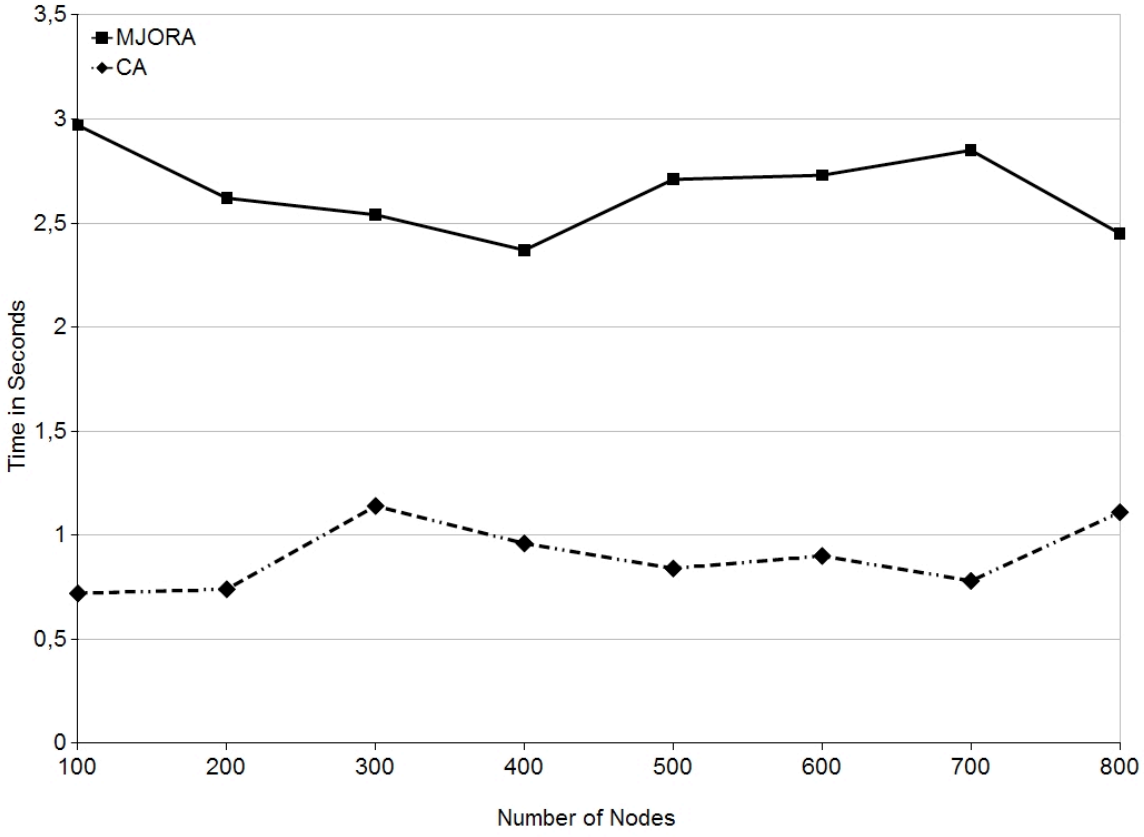

Figure 4.6 Cost of the resource allocation process for MJORA Algorithm

Like the MJORA, the CA remains also nearly constant as the number of nodes increase. This is because in CA, the algorithm stops adding new resources when the performance increase reaches to a threshold limit, instead of examining all resources in the candidate list. Otherwise, the CA would examine every resource in the grid environment. This approach may miss better resource allocation combinations with higher number of resources. However it is conceptually impossible to evaluate all possible resource allocation combinations.

Figure 4.7 shows the simulated query execution durations for the simulated query. In the figure, it can be seen that the resources allocated by the MJORA algorithm execute the query faster than the resources that are allocated by the CA. This is because, although the

resources that are allocated by the CA are the highest ranked nodes in the grid, they might be placed far from the scan nodes, which may result in slower data transfer rates. On the other hand, the resources that are allocated by the MJORA algorithm are closer to the scan nodes. For that reason, MJORA algorithm ensures allocation of more effective resources in terms of the communication performances with the scan nodes.

Regarding the simulation results, which are shown in figures 4.6 and 4.7, the MJORA algorithm is more preferable if the cost of the initial resource allocation does not exceed the estimated query execution durations. In our simulation scenarios, the durations of query executions are much higher than the costs of the resource allocation processes. Therefore, in such cases, the MJORA algorithm might be considered as a better alternative to the existing resource allocation algorithms that are based on ranking functions.

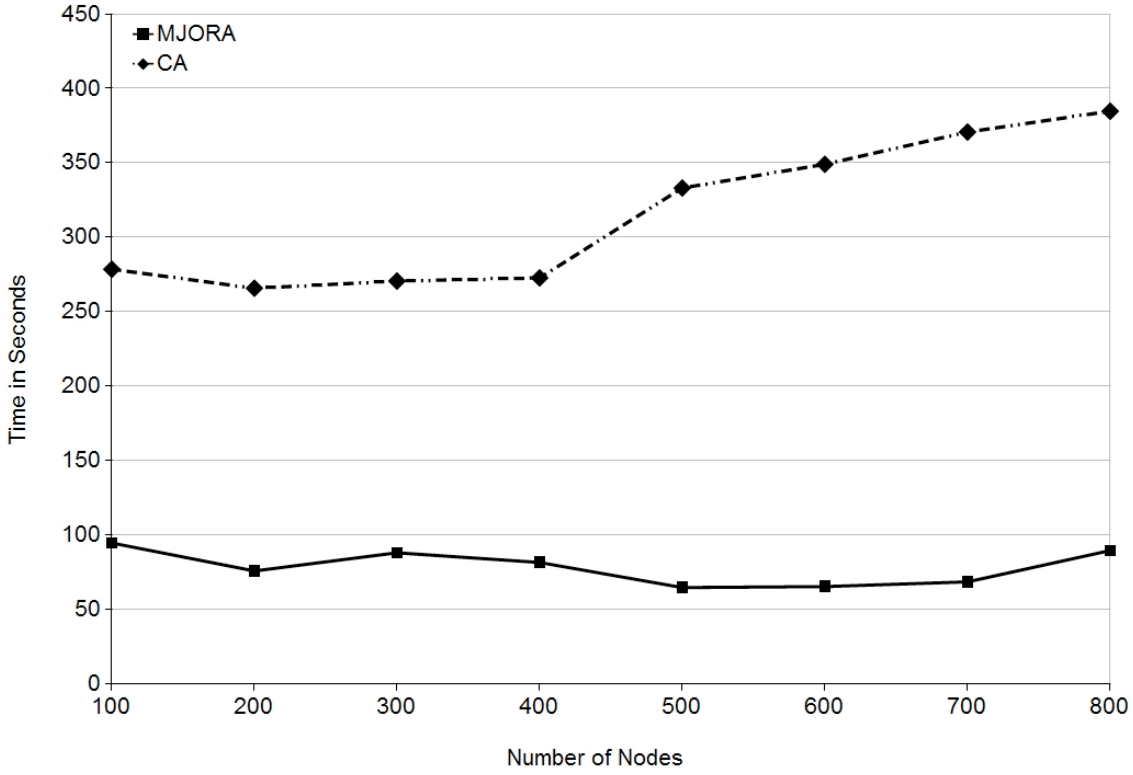

Figure 4.7 Time required to complete the query execution

4.3.4 Conclusion

In this section, we proposed the Multi-Join Resource Allocation (MJORA) algorithm, which allocates resources for a query consisting of multiple join operators by exploiting the SJORA algorithm that is presented in previous section. We presented our algorithm in detail and provided complexity analyses. We approve our analyses by the simulation and showed that our algorithm outperforms the similar existing algorithms in cases in which the query execution durations are much higher than the cost of the initial resource allocation. We showed that the MJORA algorithm can be considered a scalable algorithm that can be used as a strong alternative to the existing algorithms in many cases.

4.4 Overall Conclusions

Resource allocation is one of the key prospects that affect the performance of query processing in grid environments. Allocating right resources for tasks in queries may increase the performance drastically. However, it is proved that finding the best resource allocation is an NP-Complete problem. Therefore, spending excessive time for resource allocation step must be avoided. For this reason, current studies present heuristic approximation algorithms for resource allocation problem. The existing studies for resource allocation in grid systems were examined in detail in Chapter 2. Regarding our analyses on the current literature, we have determined the common missing points of existing resource allocation algorithms. Most of the existing resource allocation studies present detailed functions for ranking resources in the grid environment. However, we discovered that these studies rank all the resources in the grid environment, which may negatively affect the scalability of the resource allocation algorithm. Besides, we find few studies that consider the proximities of candidate resources to the data sources in the grid environment. Regarding our conclusions, we aim at contributing these common missing points of the existing studies. We first start from allocating nodes for queries that consist of single join operators considering proximities of candidate nodes to the data sources. For

this, in section 4.2, we proposed the Single Join Operator Resource Allocation (SJORA) algorithm. In SJORA, we first limit the search space of the candidate nodes using the diameter of the sub-graph consisting of data sources. Then we sort the limited candidate resource list according to the proximities of candidate nodes to the data sources. Lastly, we select nodes in this list by calculating the estimated data communication costs for the given query. We presented the complexity analyses of the proposed algorithm and consolidated these analyses with simulations. Then, by exploiting the SJORA algorithm, we proposed the Multi Join Resource Allocation (MJORA) algorithm in section 4.3, which allocates nodes for queries consisting of multiple join operators. We traverse the tree representation of the queries by using the post-order tree traversal and for each join task in the query tree, we call the SJORA algorithm to allocate nodes. We provide complexity analyses and present the simulation results of the proposed algorithm. We discussed the advantages and disadvantages of the MJORA algorithm and showed the situations in which the MJORA outperforms similar existing studies.

CHAPTER 5: FAULT TOLERANT RESOURCE ALLOCATION FOR QUERY PROCESSING IN GRID ENVIRONMENTS

Résumé

A travers ce cinquième chapitre, nous proposons un nouvel algorithme d'allocation de ressources doté d'une tolérance aux pannes pour le traitement des requêtes dans les environnements Grille. L'algorithme proposé est basé sur la réplication passive des opérateurs à état dans les requêtes. Après une présentation des analyses théoriques de l'algorithme proposé, nous consoliderons nos analyses avec les simulations.

Abstract

In this chapter, we propose a new algorithm for fault tolerant resource allocation for query processing in grid environments. The proposed algorithm is based on the passive replication of stateful operators in queries. We provide theoretical analyses of the proposed algorithm and we consolidate our analyses with the simulations.

5.1 Introduction

Grid systems are differentiated from distributed and parallel systems by their large scale, dynamic and heterogeneous characteristics [FOSTER04b]. These characteristics raise additional challenges to the distributed query processing domain such as resource discovery, resource selection, resource allocation, autonomous computing, monitoring, replication and caching, security issues and many others [GOUNARIS05]. In previous chapters, chapter 3 and chapter 4, we have proposed new algorithms for resource discovery and resource allocation problems and we have finalized resource discovery and near optimal initial resource allocation for processing queries in grid environments. However, finding near optimal resource allocation alone may not be sufficient for efficiently

processing queries in grid environments. Defining the policies in case of node failures during query execution is also very important and should be included in the resource allocation method. Since grid environments are dynamic, eventual node failures are likely during the execution of queries. These failures may be very costly if the queries are long running and if the system is not designed fault-tolerant. Therefore fault-tolerance can be considered as a must in processing queries in grid environments. Since fault-tolerance involves in allocation of new nodes, we consider it as a part of the resource allocation step.

In the current literature, there can be found many resource allocation algorithms with dynamicity support [GOUNARIS05, SILVA06, VENUGOPAL06, KOTOWSKI08, GOUNARIS09, PATON09, RUIZ09]. Although these studies provide resource allocation methods by considering dynamicity of their properties, none of them consider failure of nodes during execution of stateful operators in queries. A query operator is considered to be stateful if the execution of the operator requires storage of any kind of state such as a hash table [BESTEHORN10]. Since stateful operators like hash join, require recovery of states of the nodes in case of node failures, dynamic resource allocation methods are not sufficient in these cases. We have found few studies for fault-tolerant query processing [SMITH05, SMITH07, TAYLOR08, BESTEHORN10]. Although these studies provide fruitful algorithms, none of them is specialized on processing stateful query operators in grid environments.

In this chapter, we propose a resource allocation algorithm for fault-tolerant query processing in grid environments based on passive replication of stateful operations in queries. For finding the initial resource allocation scheme, we use the MJORA algorithm, which is proposed in chapter 4. After completion of the initial resource allocation, each allocated node applies a replication policy by itself according to the type of the task that it executes.

The contribution of this chapter is the presentation of fault-tolerance for stateful query operators in grid environments. For this, we propose Fault-Tolerant Resource Allocation (FTRA) algorithm for grid systems.

In this chapter, we assume that the node that posts the query and the nodes in which base relations reside, are fault-tolerant or stable by default during the execution of the query. We consider a query as consisting of hash join operators which are composed of atomic tasks namely scan, build and probe. Scan tasks act as providers to the build and probe tasks by reading tuples from their storage units and sending to corresponding tasks. Build tasks receive tuples from their scan tasks and create hash tables for that operator. Probe tasks are blocked during the execution of their corresponding build tasks. After build tasks complete, probe tasks start receiving tuples from their corresponding scan tasks and check for matching tuples in the hash table. Probe tasks pipeline matched tuples through their successor tasks in the query tree. More detailed description of distributed hash join operator can be found in [OZSU11].

The structure of this chapter is as follows; in section 5.2 we propose the fault-tolerant resource allocation (FTRA) algorithm for query processing in grid environments. In section 5.3 analyses of the proposed algorithm are presented. We strengthen our analyses with simulation in section 5.4. Finally, in section 5.5, we present our conclusions.

5.2 Fault-Tolerant Resource Allocation (FTRA) Algorithm

In this section, we propose the Fault-Tolerant Resource Allocation (FTRA) algorithm for query processing in grid environments based on the passive replication of stateful join operators in queries. The FTRA algorithm is responsible for the fault-tolerance of the nodes that are allocated for the join tasks. These nodes, which execute tasks, are named as master nodes and the nodes, which are allocated for the fault-tolerance purpose, are named as backup nodes or replicas. It is assumed that a master node and its replica do not fail at the same time.

5.2.1 The Algorithm

The FTRA algorithm is composed of 4 steps, *i)* replica selection, *ii)* query execution & backing-up, *iii)* failure detection and *iv)* failure recovery. These steps are defined below:

i) Replica selection: The replica selection step is realized before the master node starts its execution. When the MJORA algorithm completes, each master node determines a backup node by choosing its closest available neighbor. The selection of the closest available neighbor as the replica aims at minimizing the replication overhead that will be caused by the FTRA. In this step, we assume that a master node have always at least one available neighbor for replica selection.

ii) Query execution & backing-up: After the replica selection step, the master node starts executing the query while backing up its state for fault-tolerance. In order to avoid synchronization issues between replicas and master nodes, the replication scheme in this step is chosen to be passive replication. In passive replication, the states of the master nodes are backed-up to their replicas periodically. The states depend on the type of tasks that are being executed. More precisely, build tasks generate hash tables for the join operators. The failure of a node during the execution of build task results in the loss of the hash table that is generated so far. Therefore, during the execution of the build task the state is composed of the hash table and the sequence number of the last tuple that is received. On the other hand, probe tasks receive tuples from their predecessors, check the hash table for occurrences and send results to their successors in a pipelined fashion. Execution of a probe task means that the build task is already terminated and the hash table is already constructed. It also implies that the hash table is already backed-up. Therefore, during the execution of the probe task the state is composed only of the sequence number of the last received tuple. The update interval of the replication period is determined heuristically. The states of master nodes are backed-up incrementally. In other words, during the execution of build task, each time the state of the master node is backed-up, only the additions to the hash table is transferred to the replica node since the last back-up.

iii) Failure detection: In FTRA, the failure detection is held by both master nodes and replicas. Failures of nodes are detected by exploiting the periodical back-up messages. The replica nodes monitor failures of their master nodes and master nodes monitor failure of their replicas by examining delivery of back-up and their acknowledgement messages.

iv) Failure recovery: Whenever a replica node detects failure of its master node, it replaces itself with its master and notifies its predecessor and successor nodes for the failure recovery. Before it starts acting as a master node, it requests a replica node for itself by choosing its closest available neighbor. On the other hand, if a master node detects failure of its replica, it requests a new replica and backs-up its last state entirely once.

The algorithm is executed successively to the MJORA algorithm that is proposed in chapter 4. When the MJORA algorithm is executed beforehand, the algorithm outputs a tree structure, query tree, which contains the tree representation of the query with the selected nodes included for the initial resource allocation. A sample query tree is shown in figure 5.1.

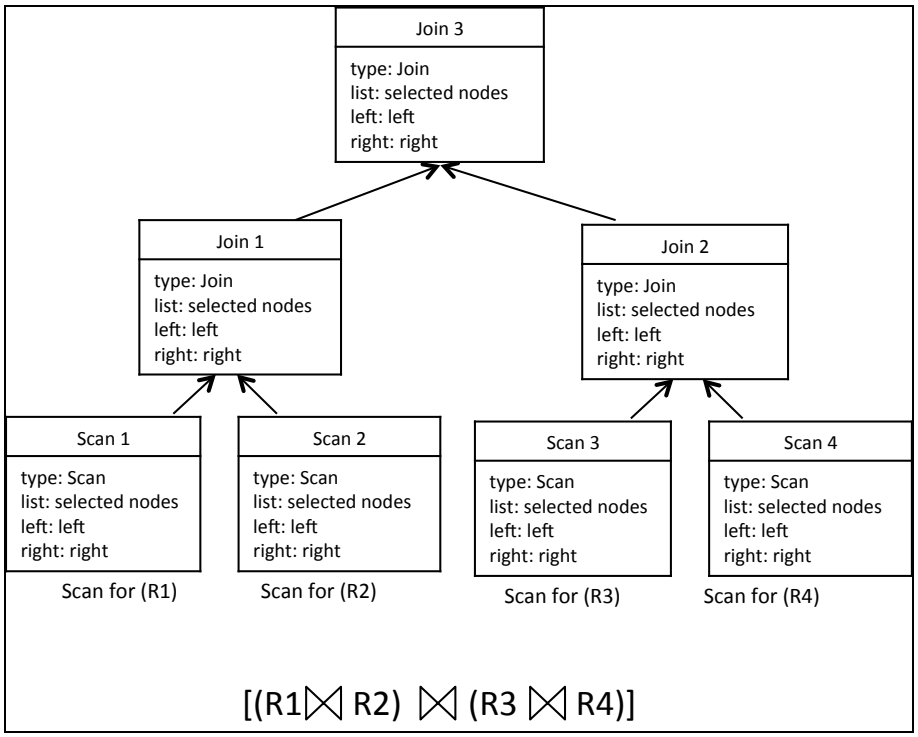

Figure 5.1 Example query tree that is used for input of the FTRA algorithm

Each vertex in the query tree is composed of a data structure named *treeElement*. The *treeElement* data structure and its contents are shown in the figure 5.2. In this data

structure, *type* represents the type of the task as whether a scan or a join task. *List* field stores the list of selected nodes for that task at the end of the MJORA algorithm. *Left* and *right* fields points to the left and right children in the query tree.

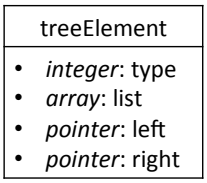

Figure 5.2 Data structure that is used in the query tree

The FTRA algorithm inputs the query tree and starts processing it by using post order tree traversal. Each time the algorithm visits a vertex in the query tree, it sends a request message to the selected resources in order to inform them about the initial allocation. This allows selected nodes to know their successor and predecessor nodes to communicate with. The formal presentation of the FTRA algorithm in the queried node is shown in algorithm 5.1.

Algorithm 5.1. FTRA algorithm in the queried node

Input: *Root of the query tree*
Output: Query execution

```
1: FTRAAlgorithm(treeNode)
2:    if treeNode is empty then
3:       return
4:    else
5:      FTRAAlgorithm(treeNode.left)
6:      FTRAAlgorithm(treeNode.right)
7:      if treeNode.type = SCAN then
8:         send AllocScanReq to the each node in treeNode.list
9:      else if treeNode.type = JOIN then
10:         send AllocJoinReq to the each node in treeNode.list
11:     end if
12:   end if
13:   return
14: end
```

After finishing the query tree traversal, the queried node waits for the resulting tuples as a pipelined fashion. The other nodes in the grid environment, which receive request messages, involve in the query execution by tracing the finite state machine steps that is

shown in Figure 5.4. Basic algorithm steps in the nodes that are assigned for different types of tasks are shown below. In algorithm 5.2, the execution of FTRA algorithm in a node that executes scan task is presented. The execution steps of the FTRA algorithm in a node that is assigned for a join task is shown in algorithm 5.3. Lastly, in algorithm 5.4 the algorithm steps for a node that is allocated for replication is presented.

Algorithm 5.2. Basic steps of FTRA algorithm in a scan node

Input: *Received messages*
Output: Execution of the required steps

1: *Upon reception of AllocScanReq message:*
2: *Wait StartScan message from all successor nodes*
3: *When all successor nodes send StartScan message*
4: *Start sending tuples to the corresponding successor nodes*
5: *When the data is completely consumed*
6: *Send ScanEnd message to all successor nodes*
7: **end**

Algorithm 5.3. Basic steps of FTRA algorithm in a join node

Input: *Received messages*
Output: Execution of the required steps

1: *Upon reception of AllocJoinReq message:*
2: *Send ReplicaReq message to the nearest neighbor*
3: *Upon receiving ReplicaResp, send StartScan message to all predecessor nodes*
4: *Build the hash table using the received tuples from the scan nodes*
5: *Periodically send hash table updates to the backup node*
6: **if** *the backup node is failed* **then**
7: *Select another backup node*
8: **end if**
9: *When build task is finished, send StartScan message to the scan nodes that hold partitions of the relation that is used in the probe task*
10: *Execute probe task using the received tuples*
11: *Periodically send last received tuple information to the backup node*
12: *When probe task is finished, send ScanEnd message to the successor nodes*
13: **end**

Algorithm 5.4. Basic steps of FTRA algorithm in a backup node

Input: *Received messages*

Output: Execution of the required steps

```
1: Upon reception of ReplicaReq message:
2:     Send ReplicaResp message to the master node
3:     Record periodical state updates
4:     if master node is failed then
5:         Send AllocUpdate message to all predecessors of the master node
           in order to suspend communication
6:         Select a backup node by sending ReplicaReq message to the
           nearest neighbor
7:        Change type of task to join
8:         Send StartScan message to all predecessors to resume the query
           execution as a master node
9:     end if
10: end
```

The finite state machine of the entire protocol for all types of tasks is shown in the Figure 5.3. The detailed description of the state machine with message types and states are described below.

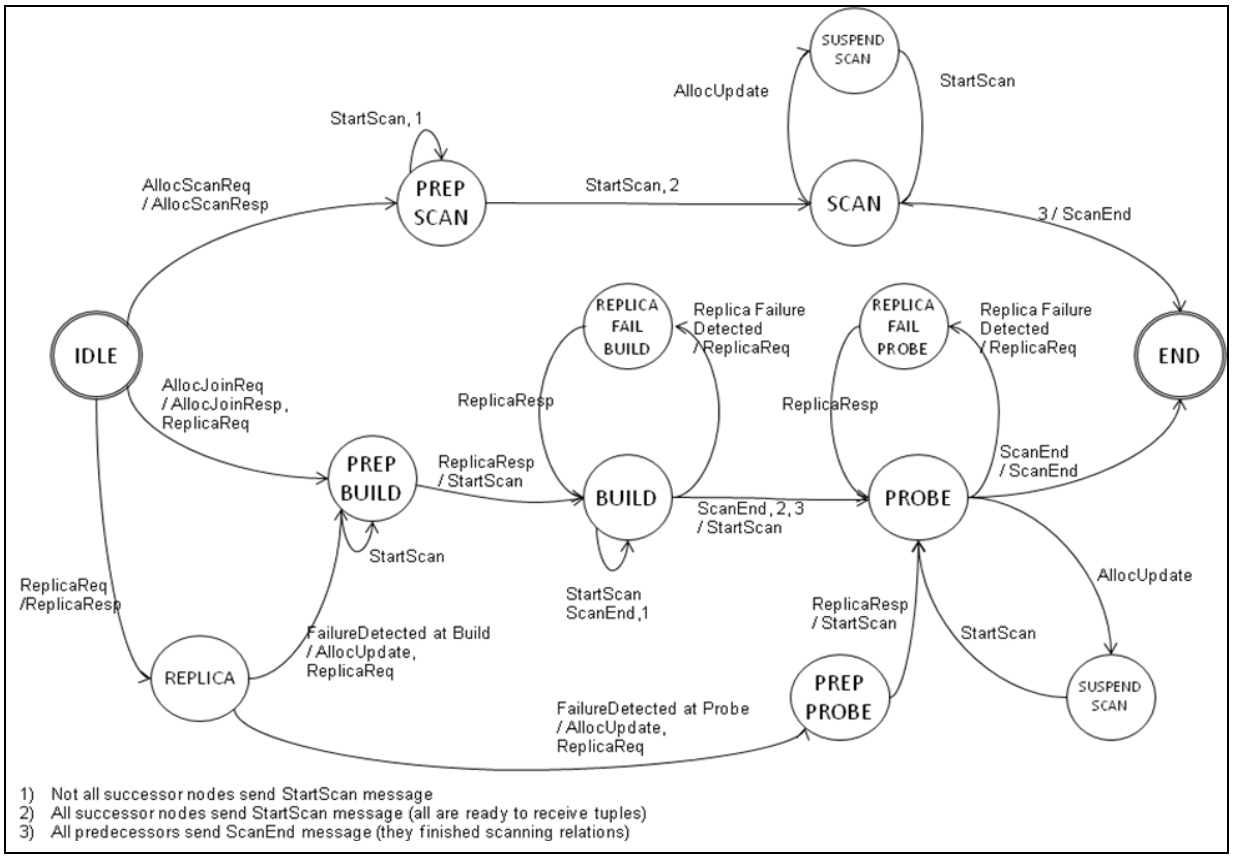

Figure 5.3 Finite state machine of the FTRA

The message types which are used in the FTRA algorithm are described below:

- *AllocScanReq:* This message is sent by the queried node to the data sources in which partitions of base relations are stored, in order to inform them for the allocation of scan tasks.
- *AllocScanResp:* The data sources accept allocation of scan tasks by sending this message to the queried node.
- *AllocJoinReq:* The nodes that are selected for join tasks are allocated by the use of this message.
- *AllocJoinResp:* The allocated nodes respond to the queried node by sending this message.

- *ReplicaReq:* Upon successful allocation of resources for the join tasks, each allocated node sends a *ReplicaReq* message to its closest neighbor in order to allocate a replica for itself.
- *ReplicaResp:* The nodes that are requested to be back-up nodes respond to their masters by sending this message.
- *StartScan:* Whenever an allocated node for a join task becomes ready for execution, it sends a *StartScan* message to its predecessor indicating that it is ready for receiving tuples.
- *AllocUpdate:* When a back-up node detects failure of its master, it becomes a master node and notifies the predecessors and successors of its failed master by sending *AllocUpdate* messages. This message also includes the checkpoint information about the execution of the task.
- *ScanEnd:* This message is sent to the successor nodes in order to notify the end of data production by either the scan or probe tasks.

During the execution of the FTRA, a node may be one of the 11 states, which are described below:

- *IDLE:* Each node starts execution of the FTRA algorithm in the *IDLE* state.
- *PREP_SCAN:* Upon reception of *AllocScanReq* message, a node changes its state to *PREP_SCAN* in order to prepare execution of scan task. Such node waits at this state until all its successors become ready for the delivery of tuples.
- *SCAN:* Whenever all successors of a scan node become ready to receive tuples, the node changes its state from *PREP_SCAN* to the *SCAN*. In this state, the node reads tuples from its storage unit and sends them to the corresponding nodes that execute join tasks.
- *SUSPEND_SCAN:* When failure of a node that executes join tasks occurs, the node which produces data for that failed node suspends producing tuples temporarily until the failed node is recovered.

- *PREP_BUILD:* Any node that is selected for a join task changes its state upon receiving *AllocJoinReq* message. Such node waits at this state until it determines a replica node before start its execution.
- *BUILD:* A node, which is at PREP_BUILD state, changes its state to BUILD whenever it receives accept response from its replica. During this state, the node receives tuples from its precedent node and generates hash table for the related partition of the relation.
- *PROBE:* When a node finishes generation of the hash table at *BUILD* state, it switches to the *PROBE* state and sends *StartScan* message to the precedents that are responsible from the partitions of probe relation. During this state, the node receives tuples from its precedents processes the join by using the hash table. The results are passed to successor nodes in a pipelined fashion.
- *REPLICA_FAIL_BUILD and REPLICA_FAIL_PROBE:* If the replica of a master node failed at the *BUILD* or *PROBE* states, the node changes its state to these states and stays there until it finds a new replica. Upon finding the new replica, the node backs-up its state on the new replica and continues its operation.
- *REPLICA:* Any node in the grid may receive requests for becoming a replica of a master node upon receiving *ReplicaReq* message. Such node replies the request if it is at the *IDLE* state. A node, which is at other states, does not reply to this request. For simplicity, we do not show such cases in the state machine. We assume that there always exist some nodes at the *IDLE* state within a node's neighborhood. During the *REPLICA* state, the node backs-up the state of its master node in periodical intervals.
- *PREP_PROBE:* When a replica detects failure of its master while the master node is in the *PROBE* state, it changes its state to *PREP_PROBE* and searches a replica node for itself. Upon finding a replica, such node changes its state to *PROBE* and continues the join task.

- *END:* Every node that executes FTRA finishes the algorithm at this state.

5.2.2 Analysis

In this section, we present time and message complexity analyses of the FTRA algorithm.

Theorem 1. *The overhead of the FTRA algorithm to the standard query execution in terms of time complexity is O(j), where j is the number of join operators in the query.*

Proof. The standard execution of a query in a grid environment consists of execution of scan, build and probe tasks. In FTRA algorithm, additionally to the standard execution, replica selection is required for each join operator before starting the execution. For each join operator, selection of a replica node involves *4* steps. Since there are *j* join operators in the query, the total time overhead of the FTRA algorithm is *(4j) which can be defined as O(j) in the big o notation*.

Theorem 1. *The overhead of the FTRA algorithm to the standard query execution in terms of message complexity is O(jnm), where j is the number of join operators in the query, n is the number of scan nodes and m is the number of allocated nodes for the join tasks.*

Proof. In FTRA algorithm, since allocated nodes become ready to receive tuples from their precedent nodes only after they set their backup nodes, beginning of scan tasks are subject to synchronization with their successors. This synchronization requires each allocated node to send a message to its scan nodes. Therefore, for each join operator, the FTRA algorithm requires *nm* message exchanges. Therefore, the total message overhead of the FTRA algorithm is *O(jnm)*.

5.2.3 Simulations

In this section we present the simulation results and quantitative evaluations for the Fault-tolerant Resource Allocation (FTRA) algorithm. We compare our algorithm to the MJORA algorithm and examine the overhead that is caused by the replication.

For the simulation scenario, we used the same simulation setting that is used in the section 4.3.3. We measured the cost of the resource allocation process.

The figure 5.4 shows the costs of MJORA and FTRA algorithms. As it can be seen in the figure, there is a slight difference between the MJORA and the FTRA. In all different simulation scenarios, the FTRA requires a constant amount of time in order to provide fault tolerance. This overhead is caused by the selection of the replica nodes. In FTRA, after completion of the initial resource allocation, the algorithm executes the finite state machine steps that are shown in the figure 5.3. In our scenarios, since the number of join operators and number of scan nodes are constant, the overhead of the FTRA algorithm is bound by the number of allocated nodes in the initial resource allocation. During our simulations, we observed that the number of allocated nodes by the MJORA algorithm does not alter significantly. Therefore, the overhead of the FTRA algorithm remains nearly constant as the number of nodes increase.

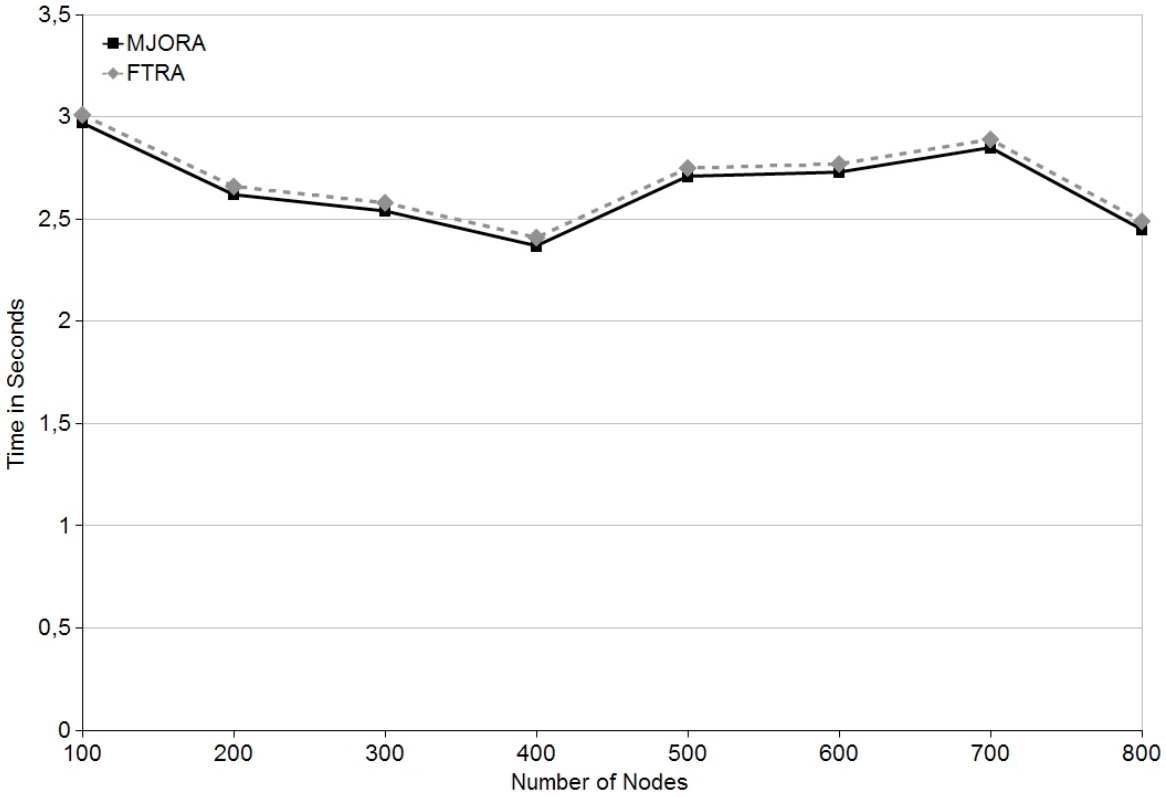

Figure 5.4 Cost of the FTRA Algorithm

5.3 Overall Conclusions

Resource allocation for query processing in grid systems is a very important step that directly affects the performance of query execution. Since grid environments are differentiated from other distributed environments by their large-scale, dynamic and heterogeneous nature, the resource allocation algorithm should address all problems that are caused by these characteristics. In previous chapter, chapter 4, we have proposed resource allocation algorithms aiming at addressing scalability and heterogeneity problems. In this chapter, aiming at addressing the dynamicity problems, we proposed a fault-tolerant resource allocation (FTRA) algorithm for query processing in grid environments that

tolerates node failures during the execution of the query. Although there can be found many studies in the current literature, to the best of our knowledge, we cannot find a study which focuses on the dynamicity of nodes in the grid environment in terms of node failures. For that reason, we aim at contributing the query-processing domain by proposing fault-tolerance in resource allocation.

In this chapter, we proposed the Fault-tolerant Resource Allocation algorithm, which ensures fault-tolerance during the execution of the query. We presented our algorithm in detail and proposed complexity analyses. Then, we strengthened our perspectives by the use of quantitative analyses and simulations. The simulation results show that the FTRA algorithm is a very favorable algorithm with little overhead to the MJORA algorithm. Since it provides fault-tolerance, it can be preferred in situations in which the queries are long running and in environments in which node failures are likely.

CHAPTER 6: CONCLUSIONS

Résumé

Dans ce chapitre final, tout en résumant les études présentées dans cette thèse, nous soulignons nos contributions et les études proposées. Ensuite, nous discutons de nouveaux problèmes qui sont soulevés par les algorithmes proposés et nous présentons nos orientations futures.

Abstract

In this chapter, we summarize studies that are presented in this thesis by highlighting our contributions and proposed studies. Then we discuss new problems that are raised by the proposed algorithms and present our future directions. Lastly, we conclude our thesis by presenting our findings and conclusions briefly.

6.1 Summary of Studies

Resource discovery and resource allocation are two of the most important research topics in distributed query processing domain, considering characteristics of grid environments. This thesis addresses these two problems by proposing new algorithms which take grid systems' characteristics into consideration. We provide these algorithms, present their analyses and consolidate the theoretical analyses with simulations. We compared the proposed algorithms with selected existing studies and highlighted our algorithms' strengths and weaknesses. The overall contributions of this thesis are listed below.

i) State-of-the-Art: We first proposed a detailed literature survey in chapter 2. We proposed evaluation criteria that we find important. Then we classified the existing studies by considering these evaluation criteria. We analyzed the literature and determined the common deficiencies of existing studies. By considering these deficiencies, we discussed the potential contributing points for each research domain,

resource discovery and resource allocation. The proposed state-of-the-art studies, which constitute the contents of chapter 2, are published in [COKUSLU09, COKUSLU10_iji, COKUSLU12, HAMEURLAIN10].

ii) Resource Discovery: After examining the current state-of-the-art on the resource discovery subject, we concluded that the resource discovery process should exploit existence of an underlying topology control mechanism. For this purpose, we proposed a suitable topology control mechanism first. Regarding the common communication requirements of resource discovery, we proposed three new self-stabilizing spanning tree construction algorithms. The constructed spanning trees are aimed to be rooted near the center of the graph in order to decrease the complexities of the resource discovery process. For this purpose, we use the degrees of nodes when determining the root of the constructed spanning tree. Then, we proposed a resource discovery algorithm by exploiting the generated topology. The proposed algorithm is scalable and prone to dynamicity of nodes in the system. Therefore, we contribute the resource discovery domain by addressing problems caused by the large scale, heterogeneity and dynamicity of the grid environments. We have designed, implemented, simulated and compared the proposed algorithms with similar existing studies in the current literature. We showed that the proposed spanning tree construction algorithms construct spanning trees with smaller diameters. We also showed that they outperform similar studies in terms of runtime durations. For the Spanning Tree Based Resource Discovery algorithm, we compared the proposed algorithm with the algorithms that use flooding during the resource discovery phase. We showed that the proposed algorithm outperforms these studies especially in large-scale environments. We also demonstrated that the proposed algorithms are scalable as the number of nodes in the grid environment increases. The contents of this chapter are partially published in [COKUSLU10].

iii) Resource allocation: After completing the resource discovery step, we have directed our attention to the resource allocation problem in chapter 4. Regarding our analyses

on the existing resource allocation studies, we provided resource allocation algorithms that consider proximities of candidate nodes to the data sources. For this, we first proposed an algorithm that allocates resources for a query that consists of a single join operator by taking the nodes' proximities to the data sources into consideration. Then we extended this algorithm to allocate resources for queries consisting of multi-join operators. We have designed, analyzed, simulated and compared our algorithms with similar algorithms in the current literature. We showed that the proposed algorithm results in better query execution performances in the simulated scenarios. We also discussed the causes of higher costs in the proposed resource allocation algorithm and examined the situations in which our algorithms are favorable. The contents of this chapter are partially published in [COKUSLU12_b].

iv) Fault-tolerant resource allocation: The algorithms that are proposed in chapter 4 address heterogeneity and large-scale characteristics of grid environments in allocating resources. In chapter 5, we addressed dynamicity problem by considering node failures during the execution of the queries. Considering our literature survey analyses for the fault-tolerant resource allocation, we have directed our attention to proposing a fault-tolerant resource allocation algorithm based on passive replication of stateful query operators in queries. We have designed, analyzed, simulated and compared our algorithm with our previously proposed resource allocation algorithm in order to evaluate fault-tolerance overhead. We showed advantages and disadvantages of the proposed algorithm and analyzed the situations in which our algorithm is favorable.

6.2 Future Directions

In this section, we present our future directions and discuss some of the challenges and open issues that we have encountered throughout the preparation of this thesis.

i) Challenges in resource discovery for query processing in grid environments: In section 3.2 we have proposed self-stabilizing spanning tree construction algorithms for topology control for resource discovery. During the proposition of the algorithms, we

showed that constructing the spanning tree with smaller diameters increases performance of the resource discovery process. We believe that constructing the spanning tree rooted at the center of the graph will further decrease the diameter of the resulting spanning tree. However, there are only few center finding algorithms that are directly focused on finding centers of arbitrary graphs. Therefore we believe that more studies can be conducted on this subject to further improve the performance of the resource discovery method.

ii) Challenges in resource allocation for query processing in grid environments: In section 4.2, we aimed at finding candidate nodes that are closest to the data sources for the resource allocation. However, there is not a study, to the best of our knowledge, which finds an arbitrary node in a graph G that is located at the center of a subset S of the graph G. This may allow finding better candidate resources for the resource allocation, since the proposed nodes will be the closest ones to all data sources. Moreover, finding weighted centers is more interesting subject, which is not yet examined in detail. By finding weighted centers, candidate nodes can be selected by considering weights of tasks that are waiting to be allocated. A larger task may be allocated closer to the data sources, whereas a smaller task, which will be executed in parallel to the larger tasks, may be allocated at farther candidates.

iii) Challenges in fault tolerant resource allocation for query processing in grid environments: In chapter 5 we proposed the Fault-tolerant Resource Allocation (FTRA) Algorithm, assuming that the queried node and data sources are stable during the execution of the query. Since in grid environments there is no such guarantee, fault-tolerance of these nodes should be studied explicitly. We also assumed that the master nodes have always at least one available node for replication within its neighborhood. Since this assumption may not always hold, it must be relaxed in the future studies.

We believe that starting from these open issues, more refined resource discovery and resource allocation techniques can be proposed.

6.3 Overall Conclusions

In this thesis, we first determined the distinguishing characteristics of the grid systems. Then we have listed the challenges for resource discovery and resource allocation domains caused by the characteristics of grid environments. We proposed resource discovery and resource allocation algorithms aiming at addressing these challenges. We believe that the algorithms, which are proposed in this thesis, contribute to the existing literature in many aspects and are useful for the query-processing domain in grid environments. We also believe that this thesis opens new issues and challenges for the query-processing domain, which may lead other researchers to search solutions to the aforementioned problems.

REFERENCES

[ABADI05] D. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The Design of the Borealis Stream Processing Engine," in *2nd Biennial Conference on Innovative Data Systems Research (CIDR'05)*, 2005, pp. 277-289.

[AFEK91] Y. Afek, S. Kutten, and M. Yung, "Memory-Efficient Self Stabilizing Protocols for General Networks," WDAG90: Proceedings of the 4th International Workshop on Distributed Algorithms, pp. 15-28, 1991.

[AHUJA89] M. Ahuja and Y. Zhu, "A Distributed Algorithm for Minimum Weight Spanning Trees Based on Echo Algorithms," Proceedings of the 9th International Conference on Distributed Computing Systems, pp. 2-8, 1989.

[ALPDEMIR04] M. Alpdemir, A. Mukherjee, A. Gounaris, N. Paton, P. Watson, A. Fernandes, and D. Fitzgerald, "OGSA-DQP: A Service for Distributed Querying on the Grid, Advances in Database Technology - EDBT 2004." vol. 2992, E. Bertino*, et al.*, Eds.: Springer Berlin / Heidelberg, 2004, pp. 3923-3923.

[ANDROUTSELLIS04] S. Androutsellis-theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys,* vol. 36, pp. 335-371, 2004.

[ANDRZEJAK02] A. Andrzejak and Z. Xu, "Scalable, efficient range queries for Grid information services," 2nd Int. Conf. on Peer-to-Peer Computing, P2P 2002, 2002.

[ANTONIOLETTI05] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. P. C. Hong, B. Collins, N. Hardman, A. C. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N. W. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead, "The design and implementation of Grid database services in OGSA-DAI: Research Articles," *Concurr. Comput. : Pract. Exper.,* vol. 17, pp. 357-376, 2005.

[ANTONOIU95] G. Antonoiu and P. K. Srimani, "A Self-Stabilizing Distributed Algorithm to Construct An Arbitrary Spanning Tree of a Connected Graph," *Computers and Mathematics with Applications,* vol. 30, pp. 1-7, 1995.

[ANTONOIU97] G. Antonoiu and P. K. Srimani, "Distributed Self-Stabilizing Algorithm for Minimum Spanning Tree Construction," Euro-Par97: Proceedings of the Third International Euro-Par Conference on Parallel Processing, London, UK, pp. 480-487, 1997.

[AWERBUCH87] B. Awerbuch, "Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and related problems," Proceedings of the 9th Annual ACM Symposium on Theory of Computing, pp. 230-240, 1987.

[BAALA03] H. Baala, O. Flauzac, J. Gaber, M. Bui, and T. El-Ghazawi, "A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks," *Journal of Parallel and Distributed Computing,* vol. 63, pp. 97-104, 2003.

[BALAZINSKA08] M. Balazinska, H. Balakrishnan, S. R. Madden, and M. Stonebraker, "Fault-tolerance in the borealis distributed stream processing system," *ACM Trans. Database Syst.,* vol. 33, pp. 1-44, 2008.

[BEHESHTI07] S. M. R. Beheshti and M. S. Moshkenani, "Development of Grid resource discovery service based on semantic information," 2007, pp. 141-148.

[BELL03] W. Bell, D. Cameron, P. Millar, L. Capozza, K. Stockinger, and F. Zini, "Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies," *International Journal of High Performance Computing Applications\,* vol. 17\, pp. 403\-416\, 2003.

[BESTEHORN10] M. Bestehorn, C. von der Weth, E. Buchmann, and K. Böhm, "Fault-tolerant query processing in structured P2P-systems," *Distributed and Parallel Databases,* vol. 28, pp. 33-66, 2010.

[BIZARRO09] P. Bizarro, N. Bruno, and D. J. DeWitt, "Progressive Parametric Query Optimization," *IEEE Trans. on Knowl. and Data Eng.,* vol. 21, pp. 582-594, 2009.

[BLIN09] L. Blin, M. G. Potop-Butucaru, and S. Rovedakis, "Self-stabilizing minimum-degree spanning tree within one from the optimal degree," IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel\&Distributed Processing, pp. 1-11, 2009.

[BOSE07] S. K. Bose, S. Krishnamoorthy, and N. Ranade, "Allocating Resources to Parallel Query Plans in Data Grids," GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing, pp. 210--220, 2007.

[BUTELLE95] F. Butelle, C. Lavault, and M. Bui, "A Uniform Self-Stabilizing Minimum Diameter Tree Algorithm (Extended Abstract)," WDAG '95: Proceedings of the 9th International Workshop on Distributed Algorithms, pp. 257-272, 1995.

[BUYYA05] R. Buyya, D. Abramson, and S. Venugopal, "The Grid Economy,"

Proceedings of the IEEE, vol. 93, pp. 698-714, 2005.

[CAI03] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A multi-attribute addressable network for Grid information services," 4th Int. Workshop on Grid Computing, GRID 2003, 2003.

[CAO02] J. Cao, D. Spooner, J. D. Turner, S. Jarvis, D. J. Kerbyson, S. Saini, and G. Nudd, "Agent-Based Resource Management for Grid Computing," CCGRID02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, p. 350, 2002.

[CHANDRASEKARAN03] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "TelegraphCQ: continuous dataflow processing," Proceedings of the 2003 ACM SIGMOD international conference on Management of data, San Diego, California, pp. 668-668, 2003.

[CHEEMA05] A. S. Cheema, M. Muhammad, and I. Gupta, "Peer-to-Peer Discovery of Computational Resources for Grid Applications," GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, pp. 179-185, 2005.

[WANG90] W. Chihping, "The complexity of processing tree queries in distributed databases," in *Parallel and Distributed Processing, 1990. Proceedings of the Second IEEE Symposium on*, 1990, pp. 604-611.

[COKUSLU09] D. Cokuslu, A. Hameurlain, and K. Erciyes, "Grid resource discovery based on web services," International Conference for Internet Technology and Secured Transactions, ICITST 2009, pp. 1 -6, 2009.

[COKUSLU10] D. Cokuslu, K. Erciyes, and A. Hameurlain, "A Maximum Degree Self-Stabilizing Spanning Tree Algorithm," in *The 25th International Symposium on Computer and Information Sciences (ISCIS 2010)*, 2010, pp. 393-396.

[COKUSLU10_iji] D. Cokuslu, A. Hameurlain, and K. Erciyes, "Grid Resource Discovery Based on Centralized and Hierarchical Architectures," *International Journal for Infonomics (IJI),* vol. 3, pp. 283-292, 2010.

[COKUSLU12] D. Cokuslu, A. Hamuerlain, and K. Erciyes, "Resource Allocation for Query Processing in Grid Systems: A Survey," *International Journal of Computer Systems Science and Engineering,* vol. 27, 2012.

[COKUSLU12_b] D. Cokuslu, A. Hamuerlain, K. Erciyes, and F. Morvan, "Resource Allocation Algorithm for a Relational Join Operator in Grid Systems," in *proceedings of 16th International Database Engineering & Applications Symposium, IDEAS12*, Prague, Czech, 2012, pp. 139-145.

[COSTA08] R. L. d. C. Costa and P. Furtado, "Scheduling in Grid Databases," Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops, pp. 696-701, 2008.

[CZAJKOWSKI01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on, pp. 181-194, 2001.

[DEWITT84] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. A. Wood, "Implementation techniques for main memory database systems," *SIGMOD Rec.,* vol. 14, pp. 1-8, 1984.

[DIJKSTRA74] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Communications of the ACM,* vol. 17, pp. 643-644, 1974.

[DING05] S. Ding, J. Yuan, J. Ju, and L. Hu, "A Heuristic Algorithm for Agent-Based Grid Resource Discovery," Intl. Conf. on e-Technology, e-Commerce and e-Service, pp. 222-225, 2005.

[DOLEV93] S. Dolev, "Optimal Time Self Stabilization in Dynamic Systems (Preliminary Version)," WDAG '93: Proceedings of the 7th International Workshop on Distributed Algorithms, pp. 160-173, 1993.

[ELANSARY03] S. El-Ansary, "Title," Type Report 2003.

[ELMROTH05] E. Elmroth and J. Tordsson, "An interoperable, standards-based grid resource broker and job submission service," e-Science and Grid Computing, 2005. First International Conference on, pp. 212-220, 2005.

[EPIMAKHOV11] I. Epimakhov, A. Hameurlain, T. Dillon, and F. Morvan, "Resource Scheduling Methods for Query Optimization in Data Grid Systems, Advances in Databases and Information Systems." vol. 6909, J. Eder*, et al*., Eds.: Springer Berlin / Heidelberg, 2011, pp. 185-199.

[COKUSLU11_b] K. Erciyes, O. Dagdeviren, D. Cokuslu, O. Yılmaz, and H. Gumus,

"Modeling and Simulation Tools for Mobile Ad hoc Networks," in *Mobile Ad Hoc Networks, Current Status and Future Trends*, J. Loo*, et al*., Eds.: CRC Press, 2011.

[FALL10] K. Fall and K. Varadhan, "Title," Type Report 2010.

[BACA89] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *Software Engineering, IEEE Transactions on,* vol. 15, pp. 1427-1436, 1989.

[FILALI08] I. Filali, F. Huet, and C. Vergoni, "A Simple Cache Based Mechanism for Peer to Peer Resource Discovery in Grid Environments," CCGRID '08, Eighth IEEE International Symposium on Cluster Computing and the Grid, pp. 602-608, 2008.

[FITZGERALD97] S. Fitzgerald, I. Foster, C. Kesselman, G. V. Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations," In Proc. 6th IEEE Symp. on High Performance Distributed Computing, pp. 365-375, 1997.

[FOSTER97] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications,* vol. 11, pp. 115-128, 1997.

[FOSTER04b] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*: Morgan Kaufmann Publishers, 2004.

[GAGLIARDI02] F. Gagliardi, B. Jones, M. Reale, and S. Burke, "European DataGrid Project: Experiences of Deploying a Large Scale Testbed for E-science Applications, Performance Evaluation of Complex Systems: Techniques and Tools." vol. 2459, M. Calzarossa and S. Tucci, Eds.: Springer Berlin / Heidelberg, 2002, pp. 255-264.

[GALLAGHER83] R. G. Gallagher, P. A. Humblet, and P. M. Spira, "A Distributed Algorithm for Minimum-Weight Spanning Trees," *ACM Transactions on Programming Languages and Systems,* vol. 5, pp. 66-77, 1983.

[GAROFALAKIS97] M. N. Garofalakis and Y. E. Ioannidis, "Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources," VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases, San Francisco, CA, USA, pp. 296--305, 1997.

[GARTNER03] F. C. Gartner, "A Survey of Self-Stabilizing Spanning-Tree Construction Algorithms," Swiss Federal Institute of Technology (EPFL), School of Computer and Communication Sciences,, Technical Report IC/2003/38 June 10, 2003.

[GOMOLUCH03] J. Gomoluch and M. Schroeder, "Market-Based Resource Allocation for Grid Computing: A Model and Simulation," 2003.

[GOUNARIS04] A. Gounaris, R. Sakellariou, N. W. Paton, and A. A. A. Fernandes, "Resource Scheduling For Parallel Query Processing On Computational Grids," Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on, pp. 396-401, 2004.

[GOUNARIS05] A. Gounaris, J. Smith, N. W. Paton, R. Sakellariou, A. A. A. Fernandez, and P. Watson, "Adapting to Changing Resource Performance in Grid Query Processing," DMG, Lecture Notes in Computer Science, pp. 30-44, 2005.

[GOUNARIS06b] A. Gounaris, R. Sakellariou, N. W. Paton, and A. A. Fernandes, "A novel approach to resource scheduling for parallel query processing on computational grids," *Distrib. Parallel Databases,* vol. 19, pp. 87-106, 2006.

[GOUNARIS09] A. Gounaris, J. Smith, N. W. Paton, R. Sakellariou, A. A. Fernandes, and P. Watson, "Adaptive workload allocation in query processing in autonomous heterogeneous environments," *Distrib. Parallel Databases,* vol. 25, pp. 125-164, 2009.

[GUPTA03] S. K. Gupta and P. K. Srimani, "Self-stabilizing multicast protocols for ad hoc networks," *Journal of Parallel and Distributed Computing,* vol. 63, pp. 87-96, 2003.

[HAMEURLAIN08] A. Hameurlain, F. Morvan, and M. E. Samad, "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, 2008.

[HAMEURLAIN10] A. Hameurlain, D. Cokuslu, and K. Erciyes, "Resource discovery in grid systems: a survey," *International Journal of Metadata, Semantics and Ontologies,* vol. 5, pp. 251-263, 2010.

[HERAULT06] T. Herault, P. Lemarinier, O. Peres, L. Pilard, and J. Beauquier, "Self-stabilizing Spanning Tree Algorithm for Large Scale Systems," *Stabilization, Safety, and Security of Distributed Systems,* vol. 4280, pp. 574-575, 2006.

[HUANG03] C. Huang, Z. Wu, G. Zheng, and X. Wu, "Dart: A Framework for Grid-Based Database Resource Access and Discovery," Grid and Cooperative Computing, pp. 855-862, 2003.

[HWANG05] J.-H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and

S. Zdonik, "High-Availability Algorithms for Distributed Stream Processing," Proceedings of the 21st International Conference on Data Engineering, pp. 779-790, 2005.

[IAMNITCHI03] A. Iamnitchi and I. Foster, "Chapter 1 A Peer-to-Peer Approach to Resource Location in Grid Environments, Kluwer," 2003.

[IAMNITCHI05] A. Iamnitchi and D. Talia, "P2P computing and interaction with grids," *Future Generation Computer Systems,* vol. 21, pp. 331-332, 2005.

[IOANNIDIS97] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis, "Parametric query optimization," *The VLDB Journal,* vol. 6, pp. 132-151, 1997.

[HWANG07] Jeong-hyon Hwang, Ying Xing, and S. Zdonik, "A cooperative, self-configuring high-availability solution for stream processing," in *ICDE*, 2007.

[JIAN07] Y. Jian and Y. Liu, "The State of the Art in Grid Scheduling Systems," Proceedings of the Third International Conference on Natural Computation - Volume 04, pp. 619-623, 2007.

[JIANG07] C. Jiang, C. Wang, X. Liu, and Y. Zhao, "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, pp. 419-427, 2007.

[JUN00] K. Jun, L. Boloni, K. Palacz, and D. C. Marinescu, "Agent-Based Resource Discovery," in Proc. 9th IEEE Heterogeneous Computing Workshop, 2000.

[KAKARONTZAS06] G. Kakarontzas and I. K. Savvas, "Agent-Based Resource Discovery and Selection for Dynamic Grids," Proc of the 15th IEEE Intl. Workshops on Enabling Technologies, pp. 195-200, 2006.

[KANT05] U. Kant and D. Grosu, "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, pp. 366-371, 2005.

[KARAATA94] M. H. Karaata, S. V. Pemmaraju, S. C. Bruell, and S. Ghosh, "Self-stabilizing algorithms for finding centers and medians of trees," PODC94, Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, New York, NY, USA, p. 374, 1994.

[KAUR07] D. Kaur and J. Sengupta, "Resource Discovery in Web-Services Based Grids,"

Proceedings Of World Academy Of Science, Engineering And Technology, pp. 284-288, 2007.

[KOSOWSKI06] A. Kosowski and L. Kuszner, "A Self-stabilizing Algorithm for Finding a Spanning Tree in a Polynomial Number of Moves," *Paralel and Distributed Non-numerical Algorithms,* vol. 3911/2006, pp. 75-82, 2006.

[KOTOWSKI08] N. Kotowski, A. A. B. Lima, E. Pacitti, P. Valduriez, and M. Mattoso, "Parallel query processing for OLAP in grids," *Concurr. Comput. : Pract. Exper.,* vol. 20, pp. 2039-2048, 2008.

[KRAUTER02] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Softw. Pract. Exper.,* vol. 32, pp. 135-164, 2002.

[KSHEMKALYANI08] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*: Cambridge University Press, 2008.

[KWON08] Y. Kwon, M. Balazinska, and A. Greenberg, "Fault-tolerant stream processing using a distributed, replicated file system," *Proc. VLDB Endow.,* vol. 1, pp. 574-585, 2008.

[LI02] W. Li, Z. Xu, F. Dong, and J. Zhang, "Grid Resource Discovery Based on a Routing-Transferring Model," 2002, pp. 145-156.

[LIEN88] Y. N. Lien, "A New Node-Join-Tree Distributed Algorithm for Minimum Weight Spanning Trees," Proceedings of the 8th International Conference on Distributed Computing System, pp. 334-240, 1988.

[LIU08] S. Liu and H. A. Karimi, "Grid query optimizer to improve query processing in grids," *Future Gener. Comput. Syst.,* vol. 24, pp. 342-353, 2008.

[MANDAL05] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson, "Scheduling Strategies For Mapping Application Workflows Onto The Grid," Proceedings, 14th IEEE International Symposium on High Performance Distributed Computing, pp. 125-134, 2005.

[MANVI05] S. S. Manvi, M. N. Birje, and B. Prasad, "An Agent-based Resource Allocation Model for Grid Computing," IEEE SCC, pp. 311-314, 2005.

[MARZOLLA05] M. Marzolla, M. Mordacchini, and S. Orlando, "Resource discovery in

a dynamic Grid environment," DEXA Workshop 2005, 2005.

[MARZOLLA07] M. Marzolla, M. Mordacchini, and S. Orlando, "Peer-to-peer systems for discovering resources in a dynamic grid," *Parallel Comput.,* vol. 33, pp. 339-358, 2007.

[MASTROIANNI05] C. Mastroianni, D. Talia, and O. Verta, "A super-peer model for resource discovery services in large-scale Grids," Future Generation Computer Systems, 2005.

[MATTOSO05] M. Mattoso, G. Zimbrão, R. A. B. Lima, A. Baião, V. P. Braganholo, A. Aveleda, B. Mir, B. K. Almentero, and M. N. Costa, "ParGRES: a middleware for executing OLAP queries in parallel," COPPE/UFRJ, Technical Report ES-690, 2005.

[MOLTO07] G. Molt\'o, V. Hern\'andez, and J. M. Alonso, "A service-oriented WSRF-based architecture for metascheduling on computational Grids," *Future Generation Computing Systems,* vol. 24, pp. 317-328, 2008.

[MOLTO07] G. Molto, V. Hernandez, and J. M. Alonso, "A service-oriented WSRF-based architecture for metascheduling on computational Grids," *Future Generation Computing Systems,* vol. 24, pp. 317-328, 2008.

[OPPENHEIMER04] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Scalable Wide-Area Resource Discovery, Technical Report CSD04 -1334, University of California Berkeley," 2004.

[OZSU11] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems, Third Edition*: Springer, 2011.

[PACITTI07] E. Pacitti, P. Valduriez, and M. Mattoso, "Grid Data Management: Open Problems and New Issues," *Journal of Grid Computing,* vol. 5, pp. 273-281, 2007.

[PAN99] R. C. Pan, J. Z. Wang, and L. R. Chow, "A self-stabilizing distributed spanning tree construction algorithm with a distributed demon," *Tamsui Oxford Journal of Mathematical Sciences,* vol. 15, pp. 23-32, 1999.

[PATON09] N. W. Paton, J. Buenabad-Chavez, M. Chen, V. Raman, G. Swart, I. Narang, D. M. Yellin, and A. A. Fernandes, "Autonomic query parallelization using non-dedicated computers: an evaluation of adaptivity options," *The VLDB Journal,* vol. 18, pp. 119-140, 2009.

[PUH07] M. Puh, G. Jezic, and M. Kusek, "Multi-agent System for Resource Discovery in

Grid Network," 16th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, pp. 320-324, 2007.

[PUPPIN05] D. Puppin, S. Moncelli, R. Baraglia, N. Tonelotto, and F. Silvestri, "A Grid information service based on Peer-to-Peer," EuroPar, Springer LNCS, 2005.

[RUIZ09] J.-A. Quian\'e-Ruiz, P. Lamarre, and P. Valduriez, "A self-adaptable query allocation framework for distributed information systems," *The VLDB Journal,* vol. 18, pp. 649-674, 2009.

[RUIZ09] J.-A. Quiane-Ruiz, P. Lamarre, and P. Valduriez, "A self-adaptable query allocation framework for distributed information systems," *The VLDB Journal,* vol. 18, pp. 649-674, 2009.

[RAMOS06] T. G. Ramos and A. C. Magalhaes, "An Extensible Resource Discovery Mechanism for Grid Computing Environments," CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, pp. 115-122, 2006.

[RANJAN08] R. Ranjan, A. Harwood, and R. Buyya, "Peer-to-peer-based resource discovery in global grids: a tutorial," *Communications Surveys \& Tutorials, IEEE,* vol. 10, pp. 6-33, 2008.

[RIEDEL07] M. Riedel, B. Schuller, D. Mallmann, R. Menday, A. Streit, B. Tweddell, M. S. Memon, A. S. Memon, B. Demuth, and T. Lippert, "Web Services Interfaces and Open Standards Integration into the European UNICORE 6 Grid Middleware," EDOCW '07: Proceedings of the 2007 Eleventh International IEEE EDOC Conference Workshop, pp. 57-60, 2007.

[ROWSTRON01] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," Lecture Notes in Computer Science, pp. 329-350, 2001.

[SEDAGHAT08] M. Sedaghat, M. Othman, and M. N. Sulaiman, Information Technology, 2008. ITSim 2008. International Symposium on, pp. 1-9, 2008.

[SILVA06] V. F. V. D. Silva, M. L. Dutra, F. Porto, B. Schulze, A. C. Barbosa, and J. C. de Oliveira, "An adaptive parallel query processing middleware for the Grid: Research Articles," *Concurr. Comput. : Pract. Exper.,* vol. 18, pp. 621-634, 2006.

[SLIMANI04] Y. Slimani, F. Najjar, and N. Mami, "An Adaptive Cost Model for Distributed Query Optimization on the Grid," OTM Workshops, pp. 79-87, 2004.

[SMITH05] J. Smith and P. Watson, "Fault-tolerance in distributed query processing," in *Database Engineering and Application Symposium, 2005. IDEAS 2005. 9th International*, 2005, pp. 329-338.

[SMITH07] J. Smith and P. Watson, "Failure Recovery Alternatives in Grid-Based Distributed Query Processing: A Case Study, Knowledge and Data Management in GRIDs," D. Talia, *et al.*, Eds.: Springer US, 2007, pp. 51-63.

[SOE05] K. M. Soe, A. A. Nwe, T. N. Aung, T. T. Naing, and N. L. Thein, "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, pp. 276-281, 2005.

[SPENCE03] D. Spence and T. Harris, "Xenosearch: Distributed resource discovery in the xenoserver open platform," In Proceedings of HPDC, p. 216, 2003.

[SULISTIO08] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, "A toolkit for modelling and simulating data Grids: an extension to GridSim," *Concurr. Comput. : Pract. Exper.,* vol. 20, pp. 1591-1609, 2008.

[TALIA04] D. Talia and P. Trunfio, "Web Services for Peer-to-Peer Resource Discovery on the Grid," Proc. of the Sixth Thematic Workshop of the EU Network of Excellence DELOS, pp. 73-84, 2004.

[TALIA05] D. Talia and P. Trunfio, "Peer-to-Peer protocols and Grid services for resource discovery on Grids," *Advances in Parallel Computing,* vol. 14, pp. 83-103, 2005.

[TALIA07] D. Talia, P. Trunfio, and J. Zeng, "Peer-to-Peer Models for Resource Discovery in Large-Scale Grids: A Scalable Architecture," High Performance Computing for Computational Science - VECPAR 2006, pp. 66-78, 2007.

[TANG06] X. Tang and L. Huang, "Grid Resource Management Based on Mobile Agent," WISE Workshops, pp. 230-238, 2006.

[TANIAR08] D. Taniar, C. H. C. Leung, W. Rahayu, and S. Goel, *High Performance Parallel Database Processing and Grid Databases*: Wiley Publishing, 2008.

[TAYLOR08] N. E. Taylor, "Recovery from Node Failure in Distributed Query Processing," University of Pennsylvania Department of Computer and Information Science, MS-CIS-08-38, 2008.

[TRUNFIO07] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi, "Peer-to-Peer resource discovery in Grids: Models and systems," *Future Gener. Comput. Syst.,* vol. 23, pp. 864-878, 2007.

[VENUGOPAL06] S. Venugopal, R. Buyya, and L. Winton, "A Grid service broker for scheduling e-Science applications on global data Grids: Research Articles," *Concurr. Comput. : Pract. Exper.,* vol. 18, pp. 685-699, 2006.

[YAN07] M. Yan, Q. Yu-hui, W. Gang, and Z. Ju-Hua, "Study of Grid Resource Discovery Based on Mobile Agent," Proc. of the 3rd Intl. Conf. on Semantics, Knowledge and Grid, pp. 570-571, 2007.

[YANG03] B. Yang and H. Garcia-Molina, "Designing a super-peer network," Data Engineering, 2003. Proceedings. 19th International Conference on, pp. 49-60, 2003.

[YIN07] Y. Yin, H. Cui, and X. Chen, "The Grid Resource Discovery Method Based on Hierarchical Model," *Information Technology Journal,* vol. 6, pp. 1090-1094, 2007.

[YU03] J. Yu, S. Venugopal, and R. Buyya, "Grid Market Directory: A Web Services based Grid Service Publication Directory," Grid Computing and Distributed Systems (GRIDS) Lab, Dept. of Computer Science and Software Engineering, The University of Melbourne, 2003.

[YU06] J. Yu, C. Zhao, and Y. Pan, "Grid Resource Management Based on Mobile Agent," Proc. of the Intl. Conf. on Computational Intelligence for Modeling Control and Automation and Intl. Conf. on Intelligent Agents Web Technologies and Intl Commerce, pp. 255-256, 2006.

[ZHAO07] X. Zhao, L. Xu, and B. Wang, "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, pp. 371-376, 2007.

Resource Discovery and Allocation for Query Processing in Grid Systems

Abstract

Grid systems are today's one of the most interesting computing environments because of their large computing and storage capabilities and their availability. Many different domains profit the facilities of grid environments. Distributed query processing is one of these domains in which there exists large amounts of ongoing research to port the underlying environment from distributed and parallel systems to the grid environment.

In this thesis, we focus on resource discovery and resource allocation algorithms for query processing in grid environments. For this, we propose resource discovery algorithm for query processing in grid systems by introducing self-stabilizing topology control and converge-cast based resource discovery algorithms. Then, we propose a resource allocation algorithm, which realizes allocation of resources for single join operator queries by generating a reduced search space for the candidate nodes and by considering proximities of candidates to the data sources. We also propose another resource allocation algorithm for queries with multiple join operators. Lastly, we propose a fault-tolerant resource allocation algorithm, which provides fault-tolerance during the execution of the query by the use of passive replication of stateful operators.

The general contribution of this thesis is twofold. First, we propose a new resource discovery algorithm by considering the characteristics of the grid environments. We address scalability and dynamicity problems by constructing an efficient topology over the grid environment using the self-stabilization concept; and we deal with the heterogeneity problem by proposing the converge-cast based resource discovery algorithm. The second main contribution of this thesis is the proposition of a new resource allocation algorithm considering the characteristics of the grid environment. We tackle the scalability problem by reducing the search space for candidate resources. We decrease the communication costs during the query execution by allocating nodes closer to the data sources. And finally we deal with the dynamicity of nodes, in terms of their existence in the system, by proposing the fault-tolerant resource allocation algorithm.

Keywords: Query processing, resource discovery, self-stabilization, topology control, spanning tree, resource allocation, fault-tolerance.

Découverte et allocation des ressources pour le traitement de requêtes dans les systèmes grilles

Résumé

De nos jours, les systèmes Grille, grâce à leur importante capacité de calcul et de stockage ainsi que leur disponibilité, constituent l'un des plus intéressants environnements informatiques. Dans beaucoup de différents domaines, on constate l'utilisation fréquente des facilités que les environnements Grille procurent.

Le traitement des requêtes distribuées est l'un de ces domaines où il existe de grandes activités de recherche en cours, pour transférer l'environnement sous-jacent des systèmes distribués et parallèles à l'environnement Grille.

Dans le cadre de cette thèse, nous nous concentrons sur la découverte des ressources et des algorithmes d'allocation de ressources pour le traitement des requêtes dans les environnements Grille. Pour ce faire, nous proposons un algorithme de découverte des ressources pour le traitement des requêtes dans les systèmes Grille en introduisant le contrôle de topologie auto-stabilisant et l'algorithme de découverte des ressources dirigé par l'élection convergente. Ensuite, nous présentons un algorithme d'allocation des ressources, qui réalise l'allocation des ressources pour les requêtes d'opérateur de jointure simple par la génération d'un espace de recherche réduit pour les nœuds candidats et en tenant compte des proximités des candidats aux sources de données. Nous présentons également un autre algorithme d'allocation des ressources pour les requêtes d'opérateurs de jointure multiple. Enfin, on propose un algorithme d'allocation de ressources, qui apporte une tolérance aux pannes lors de l'exécution de la requête par l'utilisation de la réplication passive d'opérateurs à état.

La contribution générale de cette thèse est double. Premièrement, nous proposons un nouvel algorithme de découverte de ressource en tenant compte des caractéristiques des environnements Grille. Nous nous adressons également aux problèmes d'extensibilité et de dynamicité en construisant une topologie efficace sur l'environnement Grille et en utilisant le concept d'auto-stabilisation, et par la suite nous adressons le problème de l'hétérogénéité en proposant l'algorithme de découverte de ressources dirigé par l'élection convergente. La deuxième contribution de cette thèse est la proposition d'un nouvel algorithme d'allocation des ressources en tenant compte des caractéristiques de l'environnement Grille.

Nous abordons les problèmes causés par la grande échelle caractéristique en réduisant l'espace de recherche pour les ressources candidats. De ce fait nous réduisons les coûts de communication au cours de l'exécution de la requête en allouant des nœuds au plus près des sources de données. Et enfin nous traitons la dynamicité des nœuds, du point de vue de leur existence dans le système, en proposant un algorithme d'affectation des ressources avec une tolérance aux pannes.

Mots-clés: Traitement des requêtes, découverte de ressources, contrôle de topologie auto-stabilisant, arbres couvrants, allocation des ressources, tolérance aux pannes.