Resource Allocation Algorithm for a Relational Join Operator in Grid Systems

Deniz Cokuslu[1,2]
[1]Izmir Institute of Technology,
Department of Computer Engineering
Gulbahce, Urla, 35430 Izmir, Turkey
+90(232)7507866
denizcokuslu@iyte.edu.tr

Abdelkader Hameurlain[2]
[2]IRIT, Paul Sabatier University
118 Route de Narbonne, 31062
Toulouse, France
+33(0)561558248
hameur@irit.fr

Kayhan Erciyes[3]
[3]Izmir University, Gursel Aksel Bulvari,
Uckuyular, 35350 Izmir, Turkey
+90(232)2464949
kayhan.erciyes@izmir.edu.tr

Franck Morvan[2]
[2]IRIT, Paul Sabatier University
118 Route de Narbonne, 31062
Toulouse, France
+33(0)561556325
franck.morvan@irit.fr

ABSTRACT
Grid systems become very popular during the last decade because of their rapidly increasing computational capabilities. On the other hand, the advances on different domains cause enormous increase in the scale of the manipulated data. This issue augments the importance of distributed query processing and causes researchers to port their underlying environment onto the grid systems. However the dynamicity, heterogeneity and large scale characteristics of grid systems pose new problems for the distributed query processing domain. Resource allocation for query processing in grid systems is one of these problems, which attracts many researchers' attention. In this paper, we propose a new resource allocation algorithm for one relational join operator in a query considering characteristics of the grid systems. We provide theoretical analyses of the proposed algorithm and we consolidate analyses with the simulations.

Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems – *distributed databases, query processing, relational databases.*

General Terms
Algorithms, Performance, Design.

Keywords
Resource allocation, grid systems, query processing.

1. INTRODUCTION

Grid systems are today's one of the most interesting computing environments because of their large computing and storage capabilities and their availability [7]. 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 systems [1, 15, 16, 19, 23]. However, grid system's characteristics reveal many problems. Differently from the parallel and distributed systems, grid environments are characterized as large scale, heterogeneous and dynamic in their nature [7]. It is generally assumed that grid systems have very large number of resources. These resources may correspond computational resources such as CPU, memory, storage unit or network; they may be data resources, 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. The nodes in grid systems are heterogeneous and 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 [7]. From the distributed query processing point of view, these characteristics cause many problems such as resource discovery and allocation. To efficiently execute queries in the grid environment, metadata about the relations and properties of the residing nodes should be discovered beforehand. There can be found many studies in the literature which examine resource discovery for query processing in grid systems [14]. After discovering resources, suitable allocation of resources is essential for effectively executing queries. 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 [2, 10-13, 16, 17, 20, 22]. Several survey studies examine and evaluate these studies with classification [3-5]. 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

IDEAS12 2012, August 8-10, Prague [Czech Republic]
Editors: Bipin C. Desai, Jaroslav Pokorny, Jorge Bernardino

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 [2, 17]. Thus, we believe that there are still some open issues that are not mentioned completely regarding the grid systems' characteristics.

In this study, we aim at designing a resource allocation algorithm for one join operator in a query in grid systems. 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 resources. Problems, which may arise because of the dynamicity property of grids, are not covered in this study and are planned to be examined in the near future. 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 struggles with the heterogeneity of the network by selecting candidate nodes according to their proximity to the data resources 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 produce 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 study.

The rest of this paper is structured as follows. In section 2, we present a brief literature survey related to the resource allocation algorithms for query processing in grid systems. In section 3, we introduce the Join Operator Resource Allocation algorithm in detail. In section 4, we give complexity analyses of the proposed algorithm. In section 5, quantitative evaluations and simulation results are discussed. And finally in section 6, conclusions and future works are presented.

2. RELATED WORK

There can be found many studies in the literature, which examine resource allocation for query processing in grid environments [2, 10-13, 16, 17, 20, 22]. Although these studies have common objectives, they provide different solutions for different types of parallelism for query processing. The studies [2, 16, 17, 20, 22] consider three types of parallelism, independent, pipelined and partitioned parallelism in allocating resources; whereas, the studies [10-13] propose resource allocation algorithms addressing only the partitioned parallelism.

The main idea behind the first group of algorithms, in which all different types of parallelism are considered, is to divide the query into independent sub-plans and parallelize each sub-plan regarding to the partition scheme of the relations. For instance, Soe et al. [22] first analyze the query plans for execution sequence to extract inter-query parallelism. After extracting the sequence, they map operators to different nodes by considering estimated completion times. The intra-query parallelization is tackled via iterative refinements by allocating more nodes to each operator. On the other hand, Bose et al. profit the bushy tree representation of query plans [2]. They split the query tree into two segments, and realize resource allocation for each segment separately. They repeat this process until all independent sub-plans are allocated. Then they allocate nodes for each sub-plan considering intra-query parallelism. Likewise, Kotowski et al. [16] extract independent sub-plans in the query and allocate nodes for each sub-plan considering load on the nodes. Each sub-plan is then decomposed into parallelizable portions, which will be executed using different data partitions. Liu and Karimi [17] allocates nodes with the help of a ranking function which evaluates nodes in the system. They first determine independent sub-plans in the query and allocate nodes regarding their ranks. For each independent operator, they also examine intra-query parallelism. These studies provide very interesting solutions to the resource allocation problem by considering all possible types of parallelism. However, the proposed studies do not address large scale characteristic of the grid environment since the considered search space for the candidate nodes is not specified. The algorithms mention ranking of the nodes according to some criteria, but proximity of the nodes to the data resources is not mentioned in most of the studies.

The studies in which only partitioned parallelism is examined provide more focused solutions for the resource allocation problem. For instance, Gounaris et al. [10, 12] proposed a resource scheduling method for parallel query processing in computational grids examining the partitioned parallelism problem. In their algorithm, they assumed the existence of a parallel query plan. The algorithm starts its execution by an initial resource allocation by considering the data locality. Then, for each operator, it checks if the parallelism degree of that operator can be increased. The algorithm allocates new resources until performance improvement drops below a threshold value. On the other hand, in [11, 13] Gounaris et al. enhanced their previous algorithm by providing dynamic resource allocation. In the proposed algorithm, the initial resource allocation is realized by the use of the algorithm which is proposed in [10]. Then, in each allocated node, a grid query evaluation service (GQES) is created. The GQES monitors the execution of the query sub-plan and creates notifications related to the current status of the local execution. A service named *Monitoring Event Detector* is created in each site, which evaluates the notifications. The *Diagnoser* component diagnoses the inappropriate workload and proposes an enhanced workload distribution vector to the *Responder* service. The *responder* service decides whether to move the execution to another candidate or to leave in the current resource by considering the status of the executions. The main objective in this algorithm is to make all participating resources finish at the same time. In case of a decrease in the performance of an execution, which may cause load imbalance, the algorithm makes a decision and moves the execution of the query sub-plan to another suitable node starting from the latest checkpoint. The algorithms in this group provide more focused solutions for the resource allocation problem by considering only the partitioned parallelization. Although these studies address many characteristics of the grid

environments such as dynamicity and heterogeneity, they do not propose solutions addressing large scale characteristic of the environment. Moreover, the proposed studies do not provide detailed cost models, which are focused primarily on the communication costs for the selection of candidate nodes.

In both group of algorithms, the node selection is generally realized by the use of parameters such as cpu speed, bandwidth, amount of available memory, cpu utilization etc. We find few studies in the literature, which focus particularly on the communication costs during the resource allocation [2, 17]. Since in heterogeneous systems the communication speed between nodes may highly fluctuate, we believe that networking capabilities should be considered at first glance. Besides, the search space for the candidate nodes is not specified in the examined studies. Assuming that the search space for the candidate nodes is all resources in the grid, the proposed algorithms may suffer from the scalability issues in large scale environments. To the best of our knowledge, we cannot find studies that scale down the initial search space for the candidate resources. Since grid systems are large scale, we believe that such initial refinement is essential.

3. JOIN OPERATOR RESOURCE ALLOCATION (JORA) ALGORITHM

In this section, we propose a resource allocation algorithm for a single join operator in a query. 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 that reason, we first aim at reducing the search space for the candidates of *join nodes*. To realize this, we designed the *JORA* algorithm as consisting of two complementary algorithms. In this manner, we first propose *Proximity Based Candidate List Generation (PBCG) Algorithm* in Section 3.1. After determining the list of candidate nodes, we propose *Join Task Resource Allocation (JTRA)* Algorithm in Section 3.2, which determines the parallelization degree and which finalizes the allocation of resources.

3.1 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* is shown in *Algorithm 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 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 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 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 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**

3.2 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 that 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 [2, 8, 9, 17, 18, 21]. 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 [12], 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 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 duration 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 [6]. Therefore, we heuristically normalize the S_{ij} measurements beforehand as appears in *Algorithm 4*.

Algorithm 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 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 equation (1).

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

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. 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(2nN), 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 *O(2nN)*.

5. QUANTITATIVE EVALUATION

In this section we present quantitative evaluation of the Join Operator Resource Allocation (*JORA*) algorithm by comparing with a comparative algorithm (*CA*) which reflects the common properties of the recent resource allocation algorithms such as [10, 12]. The main idea behind the *CA* is very similar to the algorithm proposed by Gounaris et al. [10, 12]. 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 *JORA* and *CA* algorithms in *ns2* simulation environment and collected results for the duration of resource allocation process and duration of query execution.

5.1 Simulation Setup

We have implemented the *JORA* and comparative algorithm (*CA*) in *ns2* simulation environment. 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 which 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.

5.2 Simulation Results

We have collected test results for the duration of resource allocation process and duration of query execution. Figure 1, shows the duration for the resource allocation process. As it can be seen in Figure 1, the duration of the execution of the *JORA* algorithm is longer than the *CA*. This is because the *JORA* algorithm processes its entire candidate list to find the best possible resource allocation within its candidates.

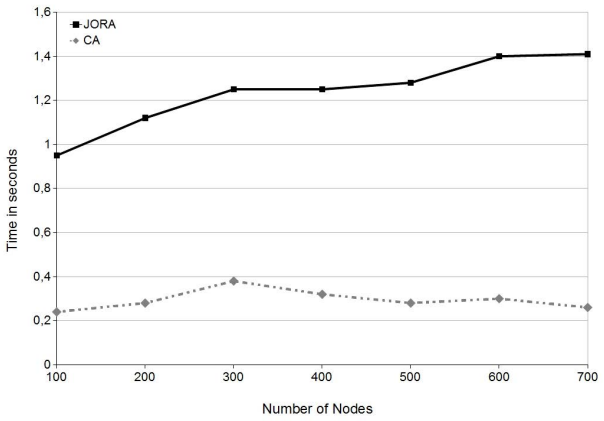

Figure 1. Duration for the resource allocation process

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 without having a limited search space.

Figure 2 shows the simulated query execution durations for the simulated query. In the figure, it can be seen that the resources allocated by the *JORA* algorithm perform better 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 *JORA* algorithm are closer to the scan nodes. For that reason, *JORA* algorithm ensures allocation of more effective resources in terms of the communication performances with the scan nodes. This heuristic causes the *JORA* algorithm outperforms the *CA*.

Regarding the simulation results which are shown in Figure 1 and 2, the *JORA* algorithm is more preferable if the resource allocation processing durations do not exceed the estimated query execution durations. In our simulation scenarios, the durations of query executions are much higher than the durations of the resource allocation processes. Therefore in such cases the *JORA* algorithm might be considered as a better alternative to the existing resource allocation algorithms which are based on ranking functions.

6. CONCLUSION

In this paper, we proposed a Join Operator Resource Allocation (*JORA*) 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 proposed 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 resources.

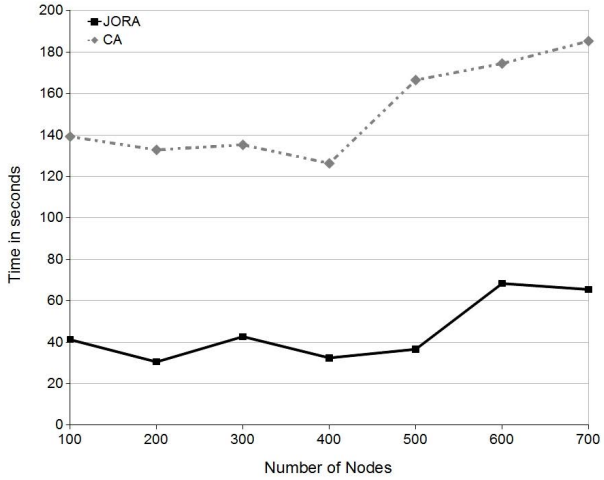

Figure 2. Query execution durations

Regarding to the results showed, we conclude that the *JORA* 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 *JORA* algorithm is a resource allocation algorithm for a single join operator in a query. For the future works, we plan to extend the *JORA* algorithm so that it will cover all join operators in an entire query. Finally we plan to mention dynamicity issues to propose a complete resource allocation algorithm for query processing in grid environments.

7. REFERENCES

[1] Antonioletti, M., Atkinson, M., Baxter, R., Borley, A., Hong, N. P. C., Collins, B., Hardman, N., Hume, A. C., Knox, A., Jackson, M., Krause, A., Laws, S., Magowan, J., Paton, N. W., Pearson, D., Sugden, T., Watson, P. and Westhead, M. The design and implementation of Grid database services in OGSA-DAI: Research Articles. *Concurr. Comput. : Pract. Exper.*, 17, 2-4 2005), 357-376.

[2] Bose, S. K., Krishnamoorthy, S. and Ranade, N. Allocating Resources to Parallel Query Plans in Data Grids. In *Proceedings of the GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing* (2007). IEEE Computer Society, [insert City of Publication],[insert 2007 of Publication].

[3] Cokuslu, D., Hamuerlain, A. and Erciyes, K. *Resource Allocation for Query Processing in Grid Systems: A Survey*. irit/rr--2010-22--fr, IRIT, Université Paul Sabatier, 2011.

[4] Costa, R. L. d. C. and Furtado, P. Scheduling in Grid Databases. In *Proceedings of the Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops* (2008). IEEE Computer Society, [insert City of Publication],[insert 2008 of Publication].

[5] Epimakhov, I., Hameurlain, A., Dillon, T. and Morvan, F. *Resource Scheduling Methods for Query Optimization in*

Data Grid Systems Advances in Databases and Information Systems. Springer Berlin / Heidelberg, City, 2011.

[6] Fall, K. and Varadhan, K. *The ns Manual (formerly ns Notes and Documentation)*. City, 2010.

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

[8] Garofalakis, M. N. and Ioannidis, Y. E. Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources. In *Proceedings of the VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1997). Morgan Kaufmann Publishers Inc., [insert City of Publication],[insert 1997 of Publication].

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

[10] Gounaris, A., Sakellariou, R., Paton, N. W. and Fernandes, A. A. A. Resource Scheduling For Parallel Query Processing On Computational Grids. In *Proceedings of the Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on* (2004), [insert City of Publication],[insert 2004 of Publication].

[11] Gounaris, A., Smith, J., Paton, N. W., Sakellariou, R., Fernandez, A. A. A. and Watson, P. Adapting to Changing Resource Performance in Grid Query Processing. In *Proceedings of the DMG, Lecture Notes in Computer Science* (2005). Springer, [insert City of Publication],[insert 2005 of Publication].

[12] Gounaris, A., Sakellariou, R., Paton, N. W. and Fernandes, A. A. A novel approach to resource scheduling for parallel query processing on computational grids. *Distrib. Parallel Databases*, 19, 2-3 2006), 87-106.

[13] Gounaris, A., Smith, J., Paton, N. W., Sakellariou, R., Fernandes, A. A. and Watson, P. Adaptive workload allocation in query processing in autonomous heterogeneous environments. *Distrib. Parallel Databases*, 25, 3 2009), 125-164.

[14] Hameurlain, A., Cokuslu, D. and Erciyes, K. Resource discovery in grid systems: a survey. *International Journal of Metadata, Semantics and Ontologies*, 5, 3 2010), 251-263.

[15] Huang, C., Wu, Z., Zheng, G. and Wu, X. Dart: A Framework for Grid-Based Database Resource Access and Discovery. In *Proceedings of the Grid and Cooperative Computing* (2003), [insert City of Publication],[insert 2003 of Publication].

[16] Kotowski, N., Lima, A. A. B., Pacitti, E., Valduriez, P. and Mattoso, M. Parallel query processing for OLAP in grids. *Concurr. Comput. : Pract. Exper.*, 20, 17 2008), 2039-2048.

[17] Liu, S. and Karimi, H. A. Grid query optimizer to improve query processing in grids. *Future Gener. Comput. Syst.*, 24, 5 2008), 342-353.

[18] Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Mellor-Crummey, J., Liu, B. and Johnsson, L. Scheduling Strategies For Mapping Application Workflows Onto The Grid. In *Proceedings of the Proceedings, 14th IEEE International Symposium on High Performance Distributed Computing* (2005), [insert City of Publication],[insert 2005 of Publication].

[19] Pacitti, E., Valduriez, P. and Mattoso, M. Grid Data Management: Open Problems and New Issues. *Journal of Grid Computing*, 5, 3 2007), 273-281.

[20] Silva, V. F. V. D., Dutra, M. L., Porto, F., Schulze, B., Barbosa, A. C. and de Oliveira, J. C. An adaptive parallel query processing middleware for the Grid: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18, 6 2006), 621-634.

[21] Slimani, Y., Najjar, F. and Mami, N. An Adaptive Cost Model for Distributed Query Optimization on the Grid. In *Proceedings of the OTM Workshops* (2004), [insert City of Publication],[insert 2004 of Publication].

[22] Soe, K. M., Nwe, A. A., Aung, T. N., Naing, T. T. and Thein, N. L. Efficient Scheduling of Resources for Parallel Query Processing on Grid-based Architecture. In *Proceedings of the Information and Telecommunication Technologies, 2005. APSITT 2005 Proceedings. 6th Asia-Pacific Symposium on* (2005), [insert City of Publication],[insert 2005 of Publication].

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