Mobile Agent-based Dynamic Resource Allocation Method for Query Optimization in Data Grid Systems

Igor Epimakhov, Abdelkader Hameurlain, Franck Morvan, Shaoyi Yin

Mobile Agent-based Dynamic Resource Allocation Method for Query Optimization in Data Grid Systems

Igor EPIMAKHOV[a], Abdelkader HAMEURLAIN[a], Franck MORVAN[a] and Shaoyi YIN[a,1]

[a] *Institut de Recherche en Informatique de Toulouse IRIT, Paul Sabatier University, France*

Abstract. Resource allocation is one of the principal stages of query processing in relational data grid systems. Specific characteristics of the data grid environment, such as dynamicity, heterogeneity and large scale, impose serious restrictions to the resource allocation process. Static resource allocation before the query execution may be far from optimal due to the dynamic changes of the system. One possible optimization is to adjust dynamically the allocation of resources during the query execution. Some methods of dynamic resource allocation have been proposed, however, most of them use centralized control mechanisms. In this study we argue that the decentralized approach meets better the requirements of the data grid systems. In this study we propose a decentralized method of dynamic resource allocation that is based on the mobile agent paradigm. We consider the participating nodes as autonomous and independent elements of the system, each of which can detect if it is overloaded and make the decision to react. Then we consider each relational operation as a mobile agent running on the allocated node, meaning that, it keeps track of its own status and can migrate to another node at any time. A two-level cooperation mechanism between such autonomous nodes and autonomous operations is described in detail. Performance evaluation proves the efficiency of the proposed method.

Keywords. Data grid systems, distributed query processing, optimization, resource allocation, load balancing, scheduling.

1. Introduction

Resource allocation is one of the principal stages of query processing in relational data grid systems. At a given time, the system containing N nodes receives a query Q and the query is decomposed into M relational operations which are usually dependent tasks. The objective of the resource allocation is to assign a part of the N nodes to perform the M operations such that the response time of Q is minimized. The scientific community gave significant considerations to this problem in the last decade. The dynamic nature of data grids raises four principal problems for the resource allocation: 1) the leaving of nodes forces the system to retransfer their load to the remaining

[1] Corresponding Author: Shaoyi YIN, Institut de Recherche en Informatique de Toulouse IRIT, Paul Sabatier University, 118 Route de Narbonne, 31062 Toulouse, France; Email: shaoyi.yin@irit.fr

nodes; 2) the unpredictable overload of nodes dynamically occurs because of leaving nodes' load retransfer and because of uncoordinated query optimization; 3) the entering nodes should be used quickly in order to balance the load in the system; 4) failures of nodes may cause data loss and data inconsistence, but we leave the fault tolerance problem out of scope of the paper.

In our survey [1] we distinguished two principal strategies for dealing with the dynamic nature of a data grid. The first strategy [2, 3, 4] allocates resources for an operation after finishing the execution of the previous operation. The second strategy [5, 6, 7, 8] combines an initial resource allocation (static phase) with a run-time reallocation (dynamic phase). In the present paper we focus on the dynamic phase of the second strategy.

We analyzed a number of works on the dynamic resource allocation problem and found that most of the published methods rely on the centralized control organization [2, 3, 4, 5, 6, 7, 8, 9, 10]. However, in a large scale dynamic system, the centralized control mechanism has some fundamental shortcomings like low reliability, risk of bottleneck and scaling problem. The main objective of the present study is to propose a decentralized method of dynamic resource allocation that could react to the dynamic fluctuations of a relational data grid system.

In the decentralized design, we consider nodes as autonomous and independent elements of the system, each of which can decide whether to execute an operation or to make it leave. Implemented as mobile agents, operations also have a limited degree of autonomy, and can make decisions about its migration and execution. In order to clearly define such a system, we must resolve the following basic questions:

1. When an operation must migrate?
2. Which operation must migrate?
3. Where should it migrate?

In addition to answering these questions, we must determine an effective structure for the interaction on two levels: 1) between a node and the operations running on it; 2) between the neighboring operations within a query. To treat this problem, we propose a two-level control system that provides cooperation between autonomous nodes and autonomous operations, clearly describing functions of each side.

The rest of the paper is organized as follows. Section 1 presents an analysis and comparison of existing methods and positions our proposed method in the domain. Section 2 describes the core of our method, including the detection of node overload, the determination of the operation to migrate, the selection of the node for migrating operation and the control structure of the system. Section 3 provides the results of performance evaluation and Section 4 concludes the paper.

2. Related Work

The problem of modifying the execution plan during the execution phase in order to adapt to dynamic changes of the grid environment attracts an attention of the scientific community and entails a number of publications in the recent years. Though a large number of works devoted to the issue of resource allocation in general purpose data grids and computational grids, we found just a few studies [2, 3, 4, 5, 6, 7, 8, 10, 11, 12] that concern the problem of dynamic resource allocation in relational data grids.

The main peculiarities of the environment consist of: 1) strong dependencies between tasks that correspond to the physical query operations; 2) high degree of fragmentation and duplication of data in the system.

In our survey [1] we analyzed the domain of resource allocation in data grids. We found that among the studied works there are two principal approaches for the adaptation of query execution plan to the dynamic unstable environment of grid. The first approach is implemented in [2, 3, 4], where the scheduler allocates resources for an operation after finishing the execution of the previous one. In this case, the scheduler can optimize the allocation quality based on the information about the availability and the load of resources, collected during execution of previous operations. The problem with this approach is that its adaptation ability is limited to the moment between finishing one operation and starting the next. The second approach [5, 6, 7, 8] combines an initial resource allocation (static phase) with a continuous execution-time monitoring and reallocation of resources (dynamic phase). The present study adopts the second approach and proposes an execution-time reallocation method.

The mechanisms of dynamic resource allocation differ in objectives, structure of control, type of reaction etc. In the rest of this section, we analyze the differences and try to position our work relative to the existing proposals.

With regard to the objectives, many [5, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18] consider the load balancing problem in order to raise the efficiency of task executions, some of which [6, 7, 8, 10, 13, 15, 16, 17, 18] also have as objective the using of new resources that appear in run-time. Another objective is the fault tolerance [8, 15]. In our method we work on the first two objectives, leaving the last one out of scope of our study.

The control structure of resource reallocation is organized differently by the authors. The work [16] proposes a global hierarchical structure that resolves the load balancing problem on the intra-group, intra-region and intra-grid levels. Many other authors use a locally centralized broker for a query [5, 7, 8, 10]. In the present paper we propose a method where the control structure is completely decentralized.

Concerning the type of reaction to dynamic fluctuations in the system, methods [5, 9] balance the load between nodes by reordering operations in execution time. There are works [17, 18] that, like our method, use a mechanism of migration of operations. Others [7, 8, 10] exploit a query rescheduling performed by a centralized query broker that does not only migrate operations but also can change the level of parallelism dynamically. An important aspect of the operation migration is the detection of the moment to migrate. Most of the authors [13, 14, 15, 16, 17, 18] propose to use the shortage of allocated resources and the presence of idle resources in the system as a triggering event of migration. However, they answer the question of how to detect the shortage of resources differently. In [16] the imbalance state is defined relatively to neighbors by comparing the processing time of a single node with the average processing time of the group containing this node. Some methods [13, 14, 15, 17, 18] detect the shortage of resources by the performance degradation. We define the shortage of resource for each node by comparing its capability with the total resource requirements of all the operations running on it. Another important aspect of the operation migration is the selection of node to which an operation will migrate. Some authors define a set of candidate resources (nodes) for the allocation process. Method [17] tries to maximize the number of dependent tasks that are located to the same node. Methods [16, 19] use neighbor principle to define candidate resources. For example, method [19] identifies neighbors based on the network topology: this approach is rather

precise, but very resource consuming. We propose to combine the data locality and neighbor principles and define a set of candidate resources using the geographical proximity.

3. Dynamic Resource Allocation Method

In this section we discuss the principal elements of our method which resolve the basic questions that we posed above in the Introduction. In our design, the migration of an operation is triggered by two types of events: 1) when a node is overloaded; 2) when a node is leaving. Section 2.1 explains how to detect the overload status of a node. If a node is overloaded, it will iteratively choose some operations and ask them to migrate. Section 2.2 shows how to choose such operations. Regarding the situation of node disconnection, all the operations that are placed on the disconnecting node must be transferred from it for liberating all its resources. When an operation is chosen to migrate, it has to decide which node it will move to. Section 2.3 describes the algorithm of node selection. In the end, Section 2.4 defines a control structure which connects the above elements.

3.1. Detection of Node Overload

To determine the state of overload of a node we need to calculate the requirements of operations that are placed on the node. If the requirements exceed the available resources, then the node is overloaded and it is necessary to free a part of its resources by reducing the number of executing operations.

The total requirements of operations are determined as the sum of requirements of each operation. We calculate separately the requirements of operations for each considered resource (CPU performance R_{CPU}, I/O bandwidth $R_{I/O}$, network connection bandwidth R_{NET}) as follows (Eq. (1)):

$$R_{CPU} = \sum R_i^{CPU},\ R_{I/O} = \sum R_i^{I/O},\ R_{NET} = \sum R_i^{NET} \tag{1}$$

where R_i^{CPU}, $R_i^{I/O}$ and R_i^{NET} are the requirements in CPU performance, I/O bandwidth and network bandwidth of the i-th operation respectively.

Our criterion of overload is expressed in the equations (Eqs. (2) and (3)):

$$\sum R_i > R_{Total} \tag{2}$$

$$R_i = \left\{R_i^{CPU}, R_i^{I/O}, R_i^{NET}\right\};\ R_{Total} = \left\{R_{Total}^{CPU}, R_{Total}^{I/O}, R_{Total}^{NET}\right\} \tag{3}$$

where R_i is the need of resources for the i-th operation; R_{Total} is the total amount of resources of the node, defined by $R_{Total}^{CPU}, R_{Total}^{I/O}, R_{Total}^{NET}$ that are available CPU performance, I/O bandwidth, network bandwidth of the node respectively.

In general, each operation is working in cooperation with other operations from the same query plan in order to provide requested data to a user. The operations in a query plan are united in a tree, in which the undermost-level operations read initial relations and the upper-level operations sequentially process them to generate the final result on the topmost level. If the operations in different levels are badly synchronized, meaning that the output speed of a sending operation is much higher or much lower than the input capacity of the receiving operation, the overall performance will be degraded.

We propose a method of self-adaption of autonomous operations, in which every operation tries to keep its performance in balance with its neighbors. More precisely, the mobile agent linked to an operation monitors the performance of lower-level neighbors of that operation and calculates the necessary quantity of resources for processing their data. The main idea is that an operation needs such a quantity of resources that it could process the input data with the same speed as the speed of data transfer from the previous operations. By demanding necessary resources from a node, the mobile agents linked to the operations of a query tree optimize and balance the resources in a cascade way from bottom to top.

3.2. Determination of the Operation to Migrate

After the detection of the overloaded state of a node, the system must reduce the load by moving a part of operations to less loaded nodes. We propose an algorithm of task excluding whose objective is to choose an operation from a set of operations to remove from the node. The operation must be transferred to other node only if it allows using optimally the resources of the node and will have a gain in time for the entire set of concurrent operations.

In general, at each moment t a set of operations O = $\{o_1, o_2, ..., o_n\}$ is executing on the node. This raises the problem of determining the operation o_i from the set O, such that its migration from the current node would decrease the total execution time of the entire set of operations. We propose the following algorithm to determine o_i.

Algorithm of task excluding
INPUT: A set of operations $O = \{o_1, o_2, ..., o_n\}$

OUTPUT: Operation o_i which should be removed

BEGIN

1. **FOR** each operation $o_i \in O$ **DO**
2. Calculate T^i_{gain} and T^i_{migr} in the case of excluding o_i
3. **ENDFOR**
4. $T_{\max} = \max_{i \in O}(T^i_{gain} - T^i_{migr})$
5. **IF** $T_{\max} > 0$ **THEN** return o_i
6. **ELSE** return null value

END

where T^i_{gain} is an estimated gain for the set of operations in the case of removing the i-th operation, T^i_{migr} is an estimation of the time to migrate the i-th operation.

The migration cost of different operations should be evaluated separately. As an example, we study here only the case of Hybrid Hash Join (HHJ) [20], one of the most versatile join algorithms. HHJ can be divided into two stages: 1) in the first stage, each received relation is divided into blocks of tuples, by performing the same hash function; 2) in the second stage, blocks from one relation are joined with the respective blocks from another relation (the join is thus broken up into a series of smaller joins) and the resulting tuples are sent to the recipient node.

Migration in the first stage would actually lead to transferring all the received data and restart the operation at another site, because partially created blocks of relations cannot be reused. We define the cost of migration at this stage as: $T_{migr}(t) = D(t) / S_{local}$ where t is the amount of time that has elapsed since the beginning of the operation, D (t) is the amount of received data at that moment and S_{local} is the local speed of data transferring from the current node.

In the second stage, blocks of tuples are fully generated and can be transferred to another node without having to restart the operation. In fact, due to sorting of the blocks, it is enough to transfer only the remaining unprocessed blocks. We define the migration cost at this stage as: $T_{migr}(t) = (D - N(t)) / S_{local}$ where D is the total size of the joining relations and N(t) is the size of already processed blocks at the time t.

3.3. Selection of the Destination Node

Algorithm of node selection for operation migration analyzes a predetermined set of candidate nodes, which we denote as Migration Space (MS). It may not include all nodes of the data grid and should be limited to the most preferred nodes. For the MS of binary relational operators such as joins, we consider firstly the source nodes of used relations, and secondly nodes that are geographically close to the source nodes (nearest nodes).

We propose an algorithm that selects the node to which the operation can migrate from the current node with minimal loss of the time.

Algorithm of node selection
INPUT: Generated migration space MS with the data about available resources for each node

OUTPUT: Node for the migration of an operation

BEGIN

1. **FOR** each node $i \in MS$ **DO**
2. Calculate $T^i_{cost} = T^i_{migr2} + T^i_{exec}$ for the i-th node
3. **ENDFOR**
4. return the node with $T_{cost} = \min_{i \in MS}(T^i_{cost})$

END

Where T^i_{exec} is an estimated execution time of all the operations on node i by adding the migrating operation and T^i_{migr2} is an estimation of the time to receive the data of the migrating operation. The estimation of T^i_{migr2} is slightly more precise than the estimation of T^i_{migr} in Section 2.2, because at this step, we have more information about

the destination node, such as the network speed. So the T^i_{migr2} is defined as follows. If the migration is done during the first phase of the HHJ operation, we have Eq. (4) and if the migration is done during the second phase, we have Eq. (5), where S_{remote} is the data receiving speed of the destination node.

$$T_{migr2}(t) = D(t) / \min(S_{local}, S_{remote}) \tag{4}$$

$$T_{migr2}(t) = (D - N(t)) / \min(S_{local}, S_{remote}) \tag{5}$$

3.4. Control Structure for the Dynamic Resource Allocation System

Choosing the paradigm of mobile operators, we have abandoned a central coordinator of the query, which can reduce the reliability of the query execution in a large-scale environment. With the new approach, each operation independently controls its own execution and migrates in cases of node overcharge or disconnection. In the process of the decision making, each operation interacts directly with neighboring operations that play the roles of data sources and data recipients.

We consider the node overload as the main reason for migration. As we defined above, to analyze the load of a node, it is necessary to accumulate information from the entire set of operations that are placed on the node. The node then makes the decision to remove one of the operations from it for the common interests of all operations. To solve this problem, we propose to use a coordinator for each node, giving to it part of responsibility in the decision-making. Thus, in the decision-making process assist two main participants: the node coordinator and the mobile agent (operation). We will describe their authorities, functions and decision-making criteria.

Node coordinator has the authority to make decisions about migration of any operation placed on its node. Its main functions are: 1) collecting information from operations and analyzing it; 2) detecting the node overload status; 3) selecting an operation to remove from the node. As a criterion for decision-making, the node coordinator considers the minimization of the total execution time of all operations placed on it.

The mobile agent makes its own decision when choosing a site on which it will migrate. Its main functions are: 1) exchanging data with neighboring operations; 2) analyzing its own resource requirements; 3) selecting a node for migration. The main decision-making criterion for the mobile agent is to minimize the execution time of all operations on the destination node.

4. Performance Evaluation

We performed an experiment in order to verify the efficiency of our method in a dynamic data grid environment. After analyzing available data grid infrastructures and simulators, we found that none of them meets completely our demands. In a real data grid we cannot use a large part of the system exclusively, so it is not possible to get repeatable results because of significant influence of other tasks, performed by nodes in parallel. Also, we did not find a data grid simulator that can completely simulate an

environment of relational data grid with unstable and heterogeneous nodes, dependent operations of the query, distributed and duplicated relations.

Thus we decided to use for the experiment our own developed simulator, described in [1]. We extended it with the capability to simulate leaving and entering nodes. We implemented also mechanisms of dynamic state monitoring.

4.1. Experiment Conditions and Measured Parameters

We simulated a segment of data grid with 200 nodes that store 200 relations. Each relation contains 5 equal fragments, each of which in turn is duplicated on 4 nodes. So in average we have 20 copies of different fragments on each node. We believe that the chosen scale is sufficient for testing the behavior of our method in dynamic conditions of data grid.

For the experiment we used a set of 100 previously generated queries, each query contains 3 join operations. The system starts the execution of all queries at the same time, so there is a significant load for all components of the system.

Simulator makes an initial resource allocation individually for each query, without taking into account the placement of other queries. That is caused by the absence of a global multi-query scheduler and by the simultaneity of resource allocation processes for the queries. We chose the condition consciously, in order to verify the ability of our method to balance the load.

As measured parameters, we chose the average execution time of a query, the number of migrations during the execution, the average load of I/O subsystem, the average load of local network connections and the number of processing queries at any time. The average execution time shows the query processing efficiency of the method. We measured the average load of I/O subsystems and the average load of network connections for studying the dynamic behavior of the system in execution time. Note that, as the local network connection, we consider the local link that connects a node to a wide-area network which connects all nodes of the data grid system. Measuring the number of queries in process during the execution lets us examine the dynamicity of queries' finalization in different test conditions.

All tests passed with the same initial set of resources and the same set of queries. We tested our system with enabled (*LB*) and disabled (*noLB*) load balancing mechanism. In the disabled mode mobile operations migrate only for liberating the node that is leaving the system. Otherwise they migrate also in order to balance the load and to improve the performance. That permits us to analyze the efficiency of the proposed load balancing method and the influence of the dynamicity of environment to the main parameters of the system.

4.2. Performance Analysis

Hereafter we examine how the main parameters of the system change during the execution in different test conditions.

4.2.1. Average Execution Time

We see in Figure 1 that, comparing to the noLB mode, activation of LB decreases the average execution time by 39.1%. So we conclude that the proposed method of load balancing significantly increases the efficiency of the system.

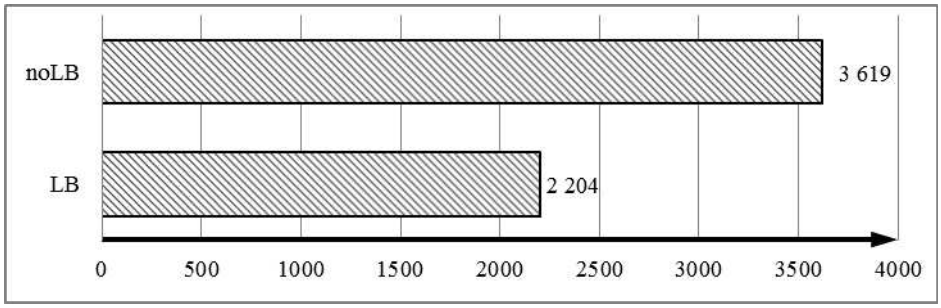

Figure 1. Average execution time of a query (seconds).

4.2.2. Number of Operation Migrations during the Query Execution

25% of the nodes leave the system at the 600th second and reenter the system at the 1200th second, as displayed in Figure 2 by dash lines. The figure shows the number of migrations initiated by the system in different periods of time. As could be seen on it, our method makes a large number of operations to migrate at the beginning. After that, there is a second large peak of migrations, which occurs at the moment of disconnecting of 25% of nodes from the data grid. In our method, all operations allocated to the disconnecting nodes are forced to migrate immediately to other nodes.

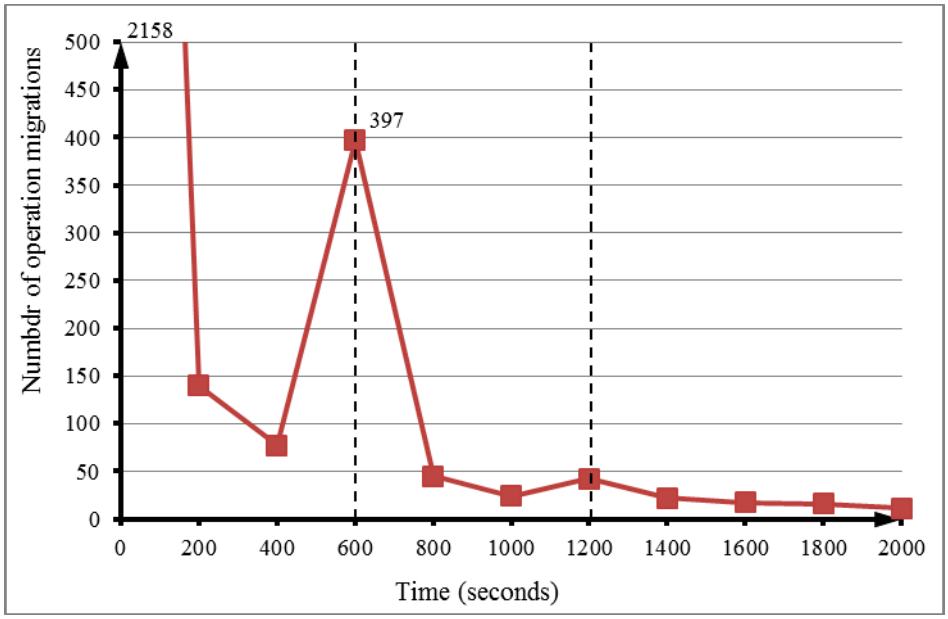

Figure 2. Number of operation migrations during the query execution.

4.2.3. Average Load of I/O Subsystem

The most obvious phenomena in Figure 3 is the large difference between LB enabled and LB disabled modes in terms of duration. The figure shows that the LB method

increases significantly the I/O subsystem utilization, which is consistent with the decreasing of the average execution time.

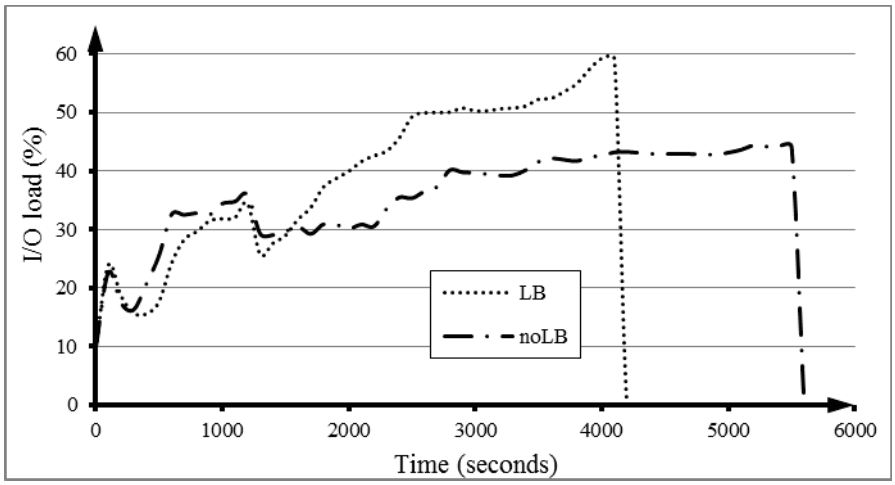

Figure 3. Average load of I/O subsystem.

4.2.4. Average Load of Network Subsystem

We can see in Figure 4 that load balancing makes higher network utilization. Analyzing the results of our experiments, we notice that higher I/O and network utilization brings lower average execution time.

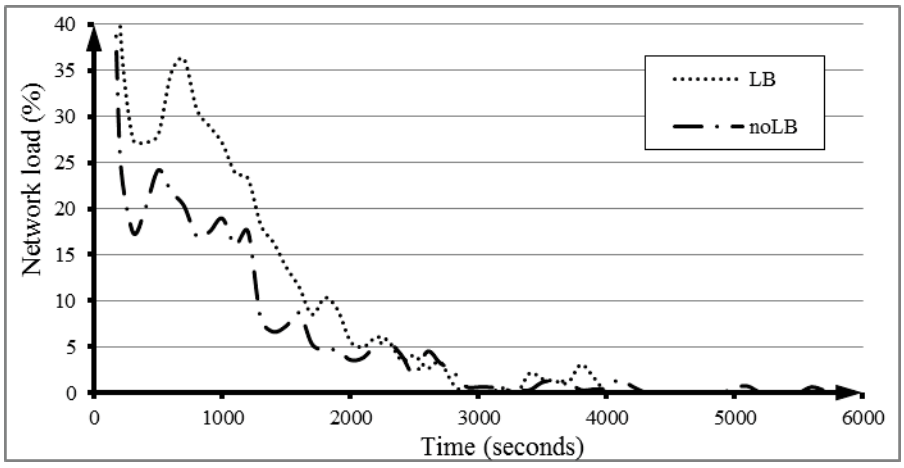

Figure 4. Average load of network subsystem.

4.2.5. Number of Queries in Process

The last diagram in Figure 5 demonstrates the dynamicity of the finalization of queries in different testing modes. The mode with enabled load balancing starts first finalizing

a bit earlier than the mode without it. Then the difference grows and becomes more significant in the end of execution than at the beginning.

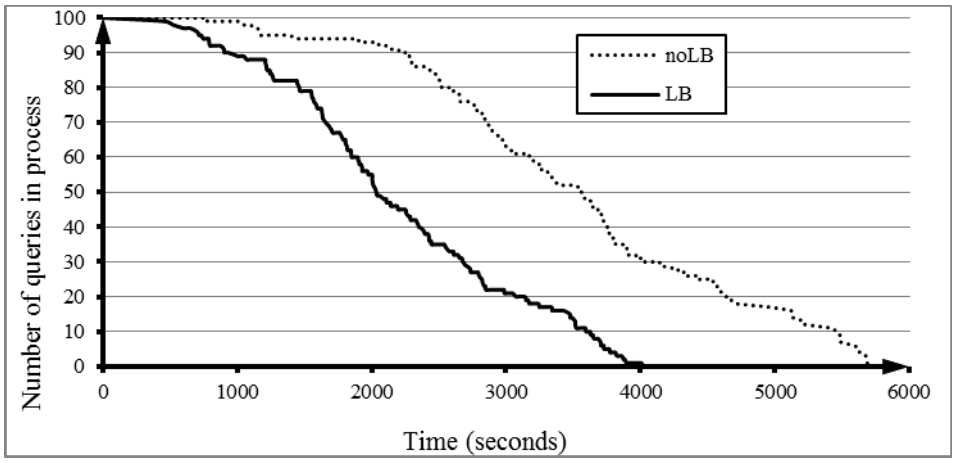

Figure 5. Number of queries in process during the query execution.

5. Conclusion

In this paper we presented a dynamic resource allocation mechanism for data grid systems. The proposed method adopts a decentralized approach, where each computing node is autonomous and can make the decision to remove operations from it. We use a mobile agent paradigm, where each operation is implemented as a mobile agent and can autonomously migrate to another node. We described the algorithms of node overload detection, migrating operation determination and destination node selection. We designed a two-level decentralized control system, which allows the cooperation between autonomous nodes and autonomous operations.

The results of performance evaluation prove the efficiency of the proposed method in terms of load balancing and nodes' instability tolerance. Comparing results with and without load balancing, we see that the proposed method increases average load of I/O and network subsystems and therefore decreases the average query execution time by 39.1%.

We conclude that the dynamic resource allocation phase during the query execution decreases efficiently the query response time in data grid systems. We believe that the decentralization of control combining with the mobile agent paradigm is a very promising approach. As the future work, we will compare our method with other related work by doing more experiments.

Acknowledgement

This work was supported in part by the French National Research Agency ANR, PAIRSE Project, Grant number -09-SEGI-008.

References

[1] I. Epimakhov, et al. Resource scheduling methods for query optimization in data grid systems. *Advances in Databases and Information Systems*, volume 6909 of *Lecture Notes in Computer Science*, 185–199. Springer Berlin / Heidelberg, 2011.

[2] H. Chen and Z. Wu. Dartgrid iii: A semantic grid toolkit for data integration. In *Proceedings of the First International Conference on Semantics, Knowledge and Grid (SKG '05),* 12–, Washington DC, 2005.

[3] H. Chen, et al. Dartgrid: a semantic infrastructure for building database grid applications: Research articles. *Concurrency Computation. : Practice & Experience*, 18(14):1811–1828, December 2006.

[4] Z. Wu, et al. Dartgrid: Semantic-based database grid. In *Proceedings of the 4th International Conference on Computational Science*, 59–66, Krakow, Poland, June 2004.

[5] V. F. V. Da Silvaet al. An adaptive parallel query processing middleware for the grid. *Concurrency and Computation: Practice and Experience*, 18(6):621–634, 2006.

[6] A. Gounaris, et al. Modular adaptive query processing for service-based grids. *CoreGRID Techinichal Report Number TR-0076*, 2007.

[7] A. Gounaris, et al. Adaptive query processing and the grid: Opportunities and challenges. In *DEXA Workshops*, 506–510, 2004.

[8] A. Gounaris, et al. Practical adaptation to changing resources in grid query processing. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 165–168, Washington DC, 2006.

[9] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the SIGMOD Conference*, 261–272, 2000.

[10] A. Gounaris, et al. Self-monitoring query execution for adaptive query processing. *Data Knowl. Eng.*, 51:325–348, December 2004.

[11] D. Cokuslu, et al. Resource allocation for query processing in grid systems: a survey. *Computer Systems: Science& Engineering*, 27(4), 2012.

[12] A. Gounaris, et al. Adaptive query processing: A survey. In *Proceedings of the 19th British National Conference on Databases (BNCOD)*, 11–25, 2002.

[13] R. Al-ali, et al. Qos adaptation in service-oriented grids. In *Proceedings of the 1st International Workshop on Middleware for Grid Computing (MGC2003) at ACM/IFIP/USENIX Middleware 2003,* Rio de Janeiro, 2003.

[14] R. Buyya, et al. A case for economy grid architecture for service oriented grid computing. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, 776 –790, April 2001.

[15] E. Huedo, et al. An experimental framework for executing applications in dynamic grid environments. *ICASE Technical Report No. 2002-43*, 2002.

[16] J. Patni, et al. Load balancing strategies for grid computing. In *Proceedings of the 3rd International Conference on Electronics Computer Technology (ICECT)*, 239 –243, April 2011.

[17] X. Sun and M. Wu. Ghs: a performance system of grid computing. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium,* Chicago, April 2005.

[18] S. Vadhiyar and J. Dongarra. A performance oriented migration framework for the grid. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003).* 130 – 137, May 2003.

[19] D. Cokuslu, et al. Resource allocation algorithm for a relational join operator in grid systems. In *Proceedings of the 16th International Database Engineering & Applications Symposium (IDEAS '12)*, 139–145, New York, 2012.

[20] D. A. Schneider and D. J. DeWitt. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment. *SIGMOD Record*, 18(2):110–121, June 1989.