QoS-Oriented Reputation-Aware Query Scheduling in Data Grids

Rogério Luís de Carvalho Costa and Pedro Furtado

University of Coimbra - Departamento de Engenharia Informática
Pólo II, Pinhal de Marrocos, 3030, 290, Coimbra, Portugal
`rogcosta@dei.uc.pt, pnf@dei.uc.pt`

Abstract. In the last few years, the Grid technology has emerged as an important tool for many scientific and commercial global organizations. In grid-based systems, intelligent job scheduling is used to achieve Service Level Objectives (SLOs) and to provide some kind of Quality of Service (QoS) differentiation between users or applications. In data grids, the grid infra-structure is used to provide transparent access to geographically distributed data, which may be replicated in order to increase availability and performance. In this work, we deal with QoS-oriented query scheduling in data grids. Although there exist several works on job scheduling in Grids, QoS-oriented query scheduling in grid-based databases is still an open issue. For instance, how can we provide guarantees against response-time expectations? Our proposal uses a reputation system to answer this problem satisfactorily. We also present experimental results that prove the benefits of proposed strategies.

1 Introduction

In the last few years, the Grid has emerged as the next generation infra-structure technology for distributed computing. It is used by a wide range of applications to provide transparent and coordinated access to distributed shared resources, including servers, workstation clusters, storage systems and databases. For instance, it can be used as basic infra-structure by global (real and virtual) organizations, which are generating huge volumes of distributed data, in order to enable geographically distributed users transparently access the distributed database. Indeed, the term *Data Grid* is commonly used to identify grid-based systems in which data is a major actor [1], including situations on which grid-based tools generate, manage or consume large volumes of data.

Grids are dynamic environments where resource availability and performance may change over time [2]. Besides that, resources are commonly heterogeneous and belong to distinct domains, which may have some degree of autonomy. In fact, local domain controllers may impose constraints on local resource utilization by remote users [2]. Moreover, in grids, job scheduling is usually QoS-oriented, which means that it aims at improving the users' satisfaction by maintaining a good Quality of Service (QoS) [3]. Service Level Objectives (SLO) are commonly specified and used to provide some kind of differentiation between jobs or users.

E. Luque, T. Margalef, and D. Benítez (Eds.): Euro-Par 2008, LNCS 5168, pp. 489–498, 2008.

In this work, we deal with QoS-oriented query scheduling in data grids, where data replication is commonly used to improve availability and performance [4]. Let's consider a situation where a query has an SLO specified in terms of an execution deadline and there are a few data services which already have the data to execute it. How to choose the best data service to execute a user's query according to its SLO and still provide high QoS-levels for other users?

One straightforward approach to schedule one query execution is to choose the data service that would finish the query execution earlier. But this may not be the best strategy. It is important to notice that the user's expectation is related to the fact that the query should be executed by its deadline and not as soon as possible. Then, when a large deadline is specified, a good local resource utilization policy may choose to execute other queries that have tighter deadlines before (or in parallel with) the one with the large deadline, in order to increase the system's SLO fulfillment level. In another scheduling strategy, one may choose to schedule the query for the slowest site between the ones that may accomplish the specified deadline. This approach aims at leaving the faster sites available to execute future queries that may have tighter deadlines than the current one. But predicting a query execution time is not simple, especially when a few queries are executed concurrently. Hence, when the prediction is wrong, the real execution time may be over the predicted time and the deadline requirement may not be achieved.

In this paper, we consider a grid configuration where the community scheduler has no total control over each site's shared resources (like in the hierarchical and decentralized scheduling models commonly used in the grid [5]). Thus, we leave to each site's controller the responsibility to estimate the query execution time at the site and to indicate if the locally available resources can execute the submitted query by the specified deadline. This is the first phase in our scheduling strategy. In the case that more than one data service candidates itself to execute the query, we use a reputation system to choose the one that should execute it. Our strategy aims at choosing the data service that has been more trustworthy in its previous commitments to accomplish specified SLOs. The use of the reputation system is the second phase of our scheduling strategy.

This work is organized as follows: in the next Section we review some related work. In Section 3 we detail the proposed reputation-based query scheduling strategy and identify some performance metrics that can be used to evaluate QoS-oriented scheduling techniques. Experimental results are presented in Section 4. Finally, we draw conclusions and discuss some future work in Section 5.

2 Related Work

In this section we review the concepts and tools involved in grid processing. We also review query scheduling over the grid and reputation systems.

Grid-based applications are commonly deployed over Grid Resource Manager GRM) Systems, like Globus Toolkit [6] and Legion [7]. Most of the GRM systems enable the use of various job scheduling policies.

The Globus Toolkit is a tools set that can be used as basic infra-structure to deploy grid applications. The available tools are related to different aspects like security, resource reservation, and data replication and movement [8]. Nimrod-G [9] and Condor-G [10] are examples of job schedulers that work over Globus.

Nimrod-G does economic-based job scheduling, scheduling a job execution for the node that has the lowest monetary cost. The GRACE (GRid Architecture for Computational Economy) middleware is used to obtain dynamic information on costs and to do auction-inspired negotiations between the scheduler and other nodes [9]. Condor-G uses the Condor's [11] ClassAds matchmaking mechanism to schedule job execution. In such strategy, jobs' requirements and nodes' capabilities are published in ClassAds (Classified Advertisements). An agent is responsible to do the matchmaking between the job's ClassAds and the available nodes' ClassAds. Globus' GRAM [12] is used to manage remote job execution.

Legion is a GRM system that aims at creating a virtual machine abstraction of the available Grid resources. In Legion, every participant is modeled as an object. Application Class objects are used to instantiate Grid applications. When instantiation an Application Class object, it is possible to specify execution requirements, including the job scheduler that should be used to schedule its execution. Legion has some built-in scheduling mechanisms (e.g. random, and round-robin [13]) but it also supports user-written job schedulers.

There are also some works on query scheduling over the grid. In [14], the authors present a distributed query processor called Polar*, that constructs distributed query execution plans in which distinct plan's operators are executed in distinct nodes. Polar* query processor is used in the OGSA-DQP strategy (Distributed Query Processor based on the Open Grid Services Architecture) [15,16]. In OGSA-DQP, web services are used to enable the use of Polar* in a grid environment. Data movement during query execution is used in order to reduce load imbalance. In [4], the authors argue that doing data movement during grid query execution may reduce performance. Hence, queries should be scheduled to nodes that already the necessary data to execute them and data replication must be done asynchronously with query execution.

The use of reputation systems to schedule job execution in grids is discussed in more recent works. In [17], the authors present generic functions that may be used to define reputation values for issues of interest and for grid service providers. The authors claim that specific equations for each context should be specified by users. In [18], reputation is used in order to detect malicious nodes which may present incomplete results for a task in donation grids. Unreliability and malicious nodes are also considered in works like [19,20].

In this work, we do not consider that a data service can provide an incomplete result for a query execution. But we consider that a local scheduler may fail to predict the necessary time to execute a query (intentionally or not), or even the predicted execution time may not be achieved due to environmental changes. Therefore, our reputation system is used to measure each service's prediction capacity and commitment degree, thus helping the global scheduler to choose the best site to execute a job according to users' expectations.

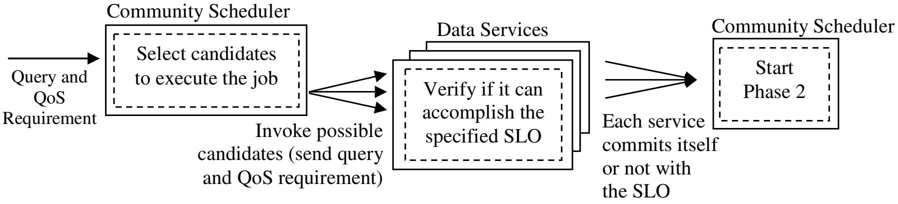

Community Scheduler
Select candidates to execute the job
Query and QoS Requirement
Invoke possible candidates (send query and QoS requirement)
Data Services
Verify if it can accomplish the specified SLO
Each service commits itself or not with the SLO
Community Scheduler
Start Phase 2

Fig. 1. QoS-oriented Reputation-aware scheduling: phase 1 main steps

3 QoS-Oriented Reputation-Based Scheduling

3.1 System Model

Grid systems may assume different architectures. We consider the existence of a *Community Query Scheduler* that may not have total control over all available shared resources in the Grid, but that is responsible to assign query execution to one of the available *Data Services*.

A Data Service is any computational resource that is capable to execute database queries, like a multi-processor database server or a cluster of workstations. If a local scheduler is used at a site, then all the resources managed by the local scheduler (and the scheduler itself) are considered by the Community Scheduler as a single Data Service. Otherwise, each resource is a Data Service that may directly interact with the Community Scheduler.

3.2 Two-Phase Reputation-Aware Scheduling Model

Our query scheduling model is divided into two phases. Initially, the services that are eligible to execute the query according to its deadline are nominated. In the second phase, a reputation model is used to select the most trustful service to execute the query.

Phase 1: Invoking Candidates

Phase 1 starts with the incoming of a new query and ends when all invoked data services answers to the Community Scheduler if they can or cannot accomplish the SLO. Figure 1 presents the main steps of Phase 1.

Selecting Candidates with the Necessary Requisites: Each submitted query is sent to the Community Scheduler, which should select the Data Services that may execute it (the ones that already have replicas of the necessary data). Such activity is supported by the use of a replica catalog, which is provided by the underlying GRM System (e.g. Globus provides a Replica Local Service that can be used by Globus-based implementations of our scheduler). Then, the Scheduler sends to each selected Data Service the query together with its QoS-requirement.

Each Selected Candidate Declares its Intention to Execute the Job: Each selected Data Service should estimate if the query can be executed with the specified SLO. It should estimate a *Local Query Execution Time* and compare the

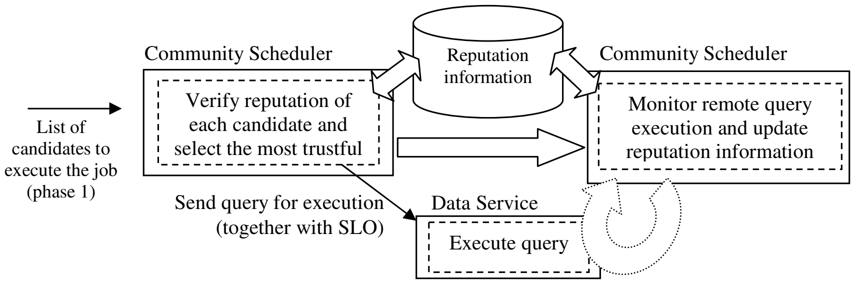

Community Scheduler
Reputation information
Community Scheduler
List of candidates to execute the job (phase 1)
Verify reputation of each candidate and select the most trustful
Monitor remote query execution and update reputation information
Send query for execution (together with SLO)
Data Service
Execute query

Fig. 2. QoS-oriented Reputation-aware scheduling: phase 2 main steps

foreseen value with the deadline requirement. Usually, the *Local Query Execution Time* includes two main factors: (i) the *Awaiting Queue Time*, which is the time that the new query would wait until other already running queries end up and its execution may begin; and (ii) the *Execution Time*, which is the necessary time to effectively execute the query against the database.

Each Data Service may have its own methods to estimate the *Local Query Execution Time*. For instance, the local query scheduler (or time estimator, when there is no local scheduler) may consider immediately starting the new query, which increases the system's multi-programming level as the new query is executed in parallel to the ones that are already executing. This would reduce the *Awaiting Queue Time* into zero, but would probably increase the *Execution Time* due to multi-query influence. Therefore, the *Execution Time* of each local scheduling alternative should be considered. Such estimation is out of this paper's scope, but some works on estimating query execution time are [21,22,23].

Phase 2: Selecting the Most Trustful Candidate

The Community Scheduler should choose one Data Service to execute the submitted query between those that had agreed to execute it by the specified SLO. In order to do that, we use a reputation system which indicates how much the Community Scheduler may trust in the Data Services' commitment to accomplish the SLO. After scheduling the query execution, the Scheduler must monitor if the query is executed by its deadline in order to update the reputation information about the selected data service. These steps are represented at Figure 2.

The Reputation Model: The reputation value we use for each Data Service is scaled to $[0, 1]$. The larger the value, the greater the confidence that the Community Scheduler has on the Data Service's commitment.

We define a *Success Factor* k ($k \in \{0, 1\}$) as an indicator if a Data Service has accomplished a deadline for a given query ($k = 1 \rightarrow$ the SLO was achieved; $k = 0 \rightarrow$ the node failed to accomplished the SLO). Then, for each Data Service (S), a reputation value R at time j is computed considering the *Success Factor* of each time (i) that the node has executed a query, against the number of times

(t) it has made itself available to execute the job. Equation 1 represents the proposed formula for R.

$$R_{S,j} = \frac{1}{\sum_{i=1}^{t} w_i} \sum_{i=1}^{t} w_i k_i \tag{1}$$

$$w_i = e^{(-\frac{\Delta t}{\lambda})} \tag{2}$$

In Equation 1, w is a time discount function used to differentiate old results from newer ones. We intend to consider newer events as more relevant than older ones. For a time window (Δt) between the time t when the query i was executed and the current time, w is computed by Equation 2 (as defined in [24]).

The parameter λ is used to allow the use of different time units and intervals, as it is defined in [24]. For instance, if the time unit used is *minute* and a *twenty minute earlier interval* should have only 10% of the effect than a new *Success Factor* value that was just obtained, then $\lambda = -\frac{20}{\ln(0.1)}$.

Updating Reputation Information: In order to update reputation information, the Community Scheduler must know if each scheduled query was finished by its deadline. Then, it monitors remote schedule execution in order to verify the job finish time. This is done with the aid of the underlying GRM infra-structure (e.g. Globus' GRAM can be used in Globus-based systems).

3.3 Performance Metrics

There are several performance metrics that can be used to describe a system performance (e.g. throughput, scalability and response time). We propose here the use of some intuitive metrics which are specific related to QoS-oriented scheduling. The proposed metrics are used in Section 4 to evaluate our scheduling model.

The first simple metric is the *SLO-Achievement Rate* (AR) of a workload. Such metric is computed considering the number of queries executed by their deadlines (S) and number of queries in the workload (Q). But some queries in the workload may have so tight deadlines that no data service commits itself to execute them by their deadlines. The *Executed Queries Rate* (EQ) is obtained considering the relation between the number of queries that were executed (N) and the number of queries in the workload (Q). The latter metric indicates the rate of queries that had at least one candidate to execute according to the desired QoS levels. The *Breach of Trust Rate* (BTR) is obtained considering the relation between the number of times a commitment to execute a query by its SLO is broken and the number of times a commitment is done (N). Equations 3, 4 and 5 represents AR, EQ and BTR, respectively.

$$AR = \frac{S}{Q} \tag{3}$$

$$EQ = \frac{N}{Q} \tag{4}$$

$$BTR = \frac{N - S}{N} \tag{5}$$

4 Experimental Results

We did several tests to validate our proposals. In this Section we present the most relevant experimental results.

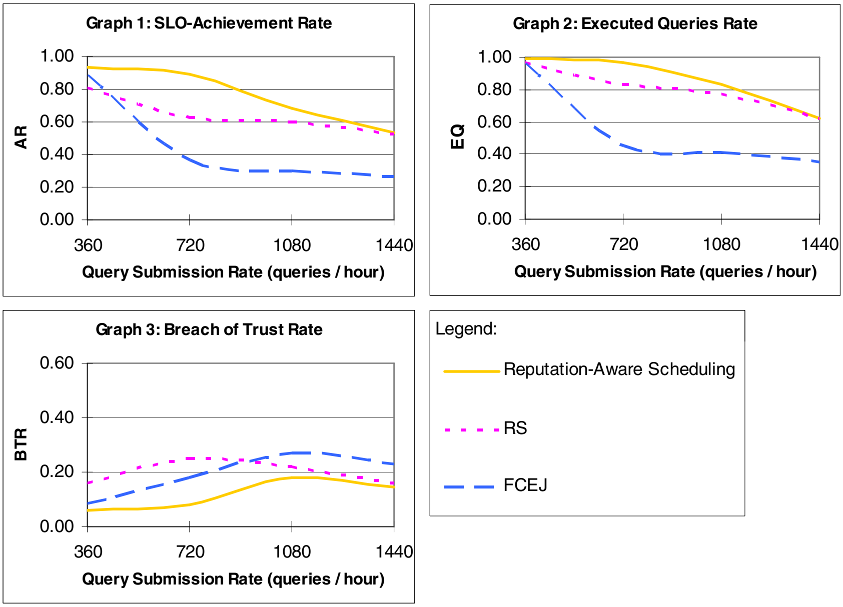
We compare our reputation-aware scheduling model with two other strategies: (i) *Random Scheduling among Candidates* (RS) and (ii) *First Candidate Executes the Job* (FCEJ). RS and FCEJ are two-phase scheduling. In their first phase, a list of candidates is generated (just like it is in our reputation-aware strategy). The main difference is in the scheduling second-phase: in RS, the query executor is randomly selected among the data services that have claimed that can execute the query according to its SLO; in FCEJ, the query is scheduled to the first data service to claim that is capable to finish query execution by its deadline. The second phase decisions of RS and FCEJ are inspired in scheduling strategies used in current GRM systems (e.g. Legion).

Our testbed workload is composed by 100 queries, that are constructed using queries 1, 2, 5, 7, 8, 9, 10, 11, 14 and 18 of TPC-H benchmark [25] with different parameters. We have simulated a data grid with six data replicas of a 1GB TPC-H database. Each replica is stored at a SQL Server 2005 DBMS placed at a different grid site. A data grid with such number of data replicas is already capable to execute a high number of concurrent queries: in our tests, we varied the query submission rate from just a few queries per minute up to 1440 queries per hour. Table 1 briefly describes some relevant hardware parameters of the experimental environment (Site 1 is used by the Community Scheduler). *Local Query Execution Time* at data services is estimated using the method proposed in [23]. Queries that have no candidates to execute them by their deadlines are aborted and do not executed (the user should re-submit the query if it is acceptable to execute the query with a larger deadline).

In the following graphs, we present the measured values for the proposed performance metrics when using the three evaluated scheduling models and different query submission rates. When using low query submission rates, the three evaluated methods achieved high values for AR (Graph 1). With low submission rates, all queries have at least one candidate to execute them (Graph 2) and even not so wise scheduling strategies can lead to high levels of QoS fulfillment. As query submission rates increase, data services start to execute several queries simultaneously

Table 1. Experimental Environment Description

Site	Processor	RAM Memory
1	Pentium IV 1.6Ghz	752MB
2	AMD Duron 1.6Ghz	752MB
3	AMD Athlon 1.5Ghz	480MB
4	AMD Athlon 1.5Ghz	736MB
5	AMD Duron 1.4Ghz	752MB
6	AMD Duron 1.6Ghz	496MB
7	Intel Xeon Dual Processor 2.8Ghz	3.87GB



and multi-query influence reduces the service's capacity to foresee the necessary time to execute a query, which leads to higher breach of trust rates (Graph 3).

In fact, the heterogeneity of our system and multi-query influence are the main factors that lead to different BTR values at each node. Multi-query influence affects in different ways the used data services. When a DBMS executes several queries concurrently, the concurrency for RAM memory space increases (especially if the accessed data tables does not fit entirely in available memory), which may lead to performance degradation (I/O performance can also suffer if different locations on disk are accessed). Thus, the smaller the RAM memory available at a data service, the greater the negative impact the service may suffer from multi-query influence.

The benefits of using the reputation-aware scheduling strategy are specially noticed when using incoming query rates between 500 and 1000 queries per hour. With submission rates in such range, the reputation-aware scheduling maintains the BTR values especially low (Graph 3), while a high number of queries is executed (Graph 2), leading to a high deadline achievement rates (Graph 1).

When using submission rates higher than 1000 queries per hour, each data service would have to execute many queries at the same time (the *Awaiting Queue Time* would have to be almost zero in order to achieve specified deadlines), which greatly increases the query execution time of each of them. Therefore, in order to avoid such high number of concurrent queries, data services deny to execute several queries and the number of queries with no candidates to execute them increases (lower values for EQ in Graph 2). This leads to lower deadline achievement rates (Graph 1), but also reduces the BTR values (Graph 3), as a smaller number of concurrent queries are effectively executed at each node (reducing multi-query influence).

Hence, the proposed reputation-aware scheduling model leads to the best deadline achievement and breach of trust rates in all studied situations.

5 Conclusions and Future Work

Globally accessible databases are becoming of great importance to a large number of real and virtual global organizations. Such environment is obtained by the use of a grid-based infra-structure, which provides transparent access to geographically distributed databases. In such Data Grids, database replication is usually done in order to increase availability and performance.

In this work, we present a new query scheduling strategy for the Data Grid. Our two-phase scheduling strategy is QoS-oriented, which means that it aims at maximizing the rate of SLO-achievement. In this work, we used execution deadlines as QoS-requirements, but our strategy may also be used for other types of QoS-requirements. The first phase of our strategy aims at selecting available sites to execute the incoming job maintains site autonomy: each site may deny to the job by the specified deadline. This can be used to implement local-domain rules. In the second phase, a reputation system to choose between the available candidates the one that should execute the query. The use of the reputation system increases the system's QoS level. We present experimental results that prove the validity of our proposals. We also identify some performance metrics that can be used to evaluate QoS-oriented scheduling techniques.

As future work, we plan to experimentally evaluate our scheduling strategy with other types of QoS-requirements and jobs.

References

1. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. J. of Network and Computer Applic. 23, 187–200 (2001)
2. Foster, I.T.: The anatomy of the grid: Enabling scalable virtual organizations. In: CCGRID, pp. 6–7 (2001)
3. Roy, A., Sander, V.: Gara: a uniform quality of service architecture. Grid resource management: state of the art and future trends, 377–394 (2004)
4. Ranganathan, K., Foster, I.: Computation scheduling and data replication algorithms for data grids. Grid resource management: state of the art and future trends, 359–373 (2004)
5. Krauter, K., Buyya, R., Maheswaran, M.: A taxonomy and survey of grid resource management systems for distributed computing. Softw. Pract. Exper. 32(2), 135–164 (2002)
6. Foster, I., Kesselman, C.: Globus: A metacomputing infrastructure toolkit. The Internat. Journal of Superc. Appl. and High Perf. Computing 11(2), 115–128 (1997)
7. Grimshaw, A.S., Wulf, W.A., Team, T.L.: The legion vision of a worldwide virtual computer. Commun. ACM 40(1), 39–45 (1997)

8. Foster, I.T.: Globus toolkit version 4: Software for service-oriented systems. J. Comput. Sci. Technol. 21(4), 513–520 (2006)
9. Buyya, R., Abramson, D., Giddy, J.: Nimrod/g: An architecture of a resource management and scheduling system in a global computational grid. CoRR cs.DC/0009021 (2000)
10. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-g: A computation management agent for multi-institutional grids. Cluster Computing 5(3), 237–246 (2002)
11. Tannenbaum, T., Wright, D., Miller, K., Livny, M.: Condor – a distributed job scheduler. In: Beowulf Cluster Computing with Linux. MIT Press, Cambridge (2001)
12. Czajkowski, K., Foster, I.T., Karonis, N.T., Kesselman, C., Martin, S., Smith, W., Tuecke, S.: A resource management architecture for metacomputing systems. In: Proc. of the Work. on Job Scheduling Strat. for Parallel Processing, pp. 62–82 (1998)
13. Natrajan, A., Humphrey, M.A., Grimshaw, A.S.: Grid resource management in legion. Grid resource manag.: state of the art and future trends, 145–160 (2004)
14. Smith, J., Gounaris, A., Watson, P., Paton, N.W., Fernandes, A.A.A., Sakellariou, R.: Distributed query processing on the grid. In: Parashar, M. (ed.) GRID 2002. LNCS, vol. 2536, pp. 279–290. Springer, Heidelberg (2002)
15. Alpdemir, N.M., Mukherjee, A., Gounaris, A., Paton, N.W., Watson, P., Fernandes, A.A., Fitzgerald, D.J.: Ogsa-dqp: A service for distributed querying on the grid. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 858–861. Springer, Heidelberg (2004)
16. Gounaris, A., Smith, J., Paton, N.W., Sakellariou, R., Fernandes, A.A.A., Watson, P.: Adapting to changing resource performance in grid query processing. In: DMG, pp. 30–44 (2005)
17. Silaghi, G., Arenas, A., Silva, L.: A utility-based reputation model for service-oriented computing. In: Proc. of the CoreGRID Symposium, pp. 63–72 (2007)
18. Sonnek, J., Nathan, M., Chandra, A., Weissman, J.: Reputation-based scheduling on unreliable distributed infrastructures. In: ICDCS 2006: Proc. of the 26th IEEE Inter. Conf. on Distributed Computing Systems, p. 30 (2006)
19. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: WWW 2003: Proc. of the 12th Inter. Conf. on World Wide Web, pp. 640–651 (2003)
20. Singh, A., Liu, L.: Trustme: Anonymous management of trust relationships in decentralized p2p systems. In: Peer-to-Peer Computing, pp. 142–149 (2003)
21. Spiliopoulou, M., Hatzopoulos, M., Vassilakis, C.: A cost model for the estimation query execution time in a parallel environment supporting pipeline. Computers and Artificial Intelligence (4) (1996)
22. Tomov, N., Dempster, E., Williams, M.H., Burger, A., Taylor, H., King, P.J.B., Broughton, P.: Analytical response time estimation in parallel relational database systems. Parallel Comput. 30(2), 249–283 (2004)
23. de Carvalho Costa, R.L., Furtado, P.: A qos-oriented external scheduler. In: SAC 2008: Proceedings of the 2008 ACM symposium on Applied computing, pp. 1029–1033. ACM, New York (2008)
24. Huynh, T.D., Jennings, N.R., Shadbolt, N.R.: An integrated trust and reputation model for open multi-agent systems. Autonomous Agents and Multi-Agent Systems 13(2), 119–154 (2006)
25. Transaction processing council benchmarks - (Last Visited in January 2008), `http://www.tpc.org/`