ADAPTIVE RESOURCES PROVISIONING FOR GRID APPLICATIONS AND SERVICES

A. Filali and A. S. Hafid
Network Research Lab
University of Montreal, Canada
Email {afilali, ahafid}@iro.umontreal.ca

M. Gendreau
CIRRELT
University of Montreal, Canada
Email: michel.gendreau@cirrelt.ca

Abstract - **Applications utilizing Grid computing infrastructure usually require resources allocation (e.g., bandwidth and CPU) to satisfy their Quality of Service (QoS) requirements. Given the dynamic nature of grid computing, Quality of Service (QoS) support and adaptation must be a high priority to successfully support those applications. In this paper, we present an adaptive resources provisioning scheme that optimizes the resources utilization while satisfying the required QoS. More specifically, it minimizes the request blocking probability and, thus, maximizes the revenues of the infrastructure provider.**

Keywords- resources provisioning, adaptation, Grid, Quality of Service, Integer programming, Heuristics

I. INTRODUCTION

Grid computing provides a global-scale distributed computing infrastructure for executing scientific and business applications [5]. Many of these applications, which have soft-real time constraints, require QoS support and assurance to execute properly. This makes it imperative to provide more stringent QoS assurances beyond those provided by the basic Grid infrastructure. In this context, a service level agreement is necessary, that specifies exactly all expectations and obligations in the business relationship between the provider and the customer [4, 8, 10]. The provider has to allocate the amount of resources, e.g., CPU and bandwidth, necessary to satisfy the agreed upon level of service. In [6], the authors present a performance tuning system that reflects changing requirements, applying real-time adaptive control technique to dynamically adapt to changing application resource demands and system resources availability. The authors in [9] present a mechanism supporting open reservations to deal with the dynamic Grid and providing a practical solution for agreement enforcement

Considerable research efforts have been dedicated to Quality of Service (QoS) management in Grid systems with particular emphasis on network resources [1,2,3,7]. Most of existing approaches, if not all, assume one QoS value for each type of resources, e.g., 2 Mbps for bandwidth and 100 CPU cycles, during the time interval [startTime, stopTime]. The authors in [1] propose a "Grid QoS Management" framework that includes activities to manage QoS, such as enabling users to specify their QoS, selection and allocation of resources according to QoS requirements, monitoring to keep track of resources availability, etc. Particularly, they propose an adaptation algorithm that reserves resources for three types of services ('Guaranteed', 'best effort', and an 'adaptive'); the algorithm tries to satisfy each new request by adjusting resources reservations between the three types of services (e.g., reducing the amount of resources reserved for a "best service" request to accommodate a "guaranteed" service request). The adjustment helps avoiding the underutilisation of the Grid resources. However, the proposed approach handles only one QoS value for each type of resource.

We believe that the utilization of resources can be considerably improved by allowing users to specify more than one value for each type of resources; indeed, this is suitable for several Grid applications for which the requirements can be satisfied using more than one value of QoS for a given resource. For example, at time currentTime, a Grid application requires the transfer of a file F (size=60 Gb) from A to B within 10 minutes (e.g., at currentTime+10, the application will have the necessary CPU cycles to process the file). This request can be satisfied using different bandwidth reservations: 1Gbps for 1 minute, 500 Mbps for 2 minutes, 250 Mbps for 4 minutes, etc. The reservation that will be selected will depend on the amount of resources available at currentTime.

In this paper, we propose an adaptive scheme that maximizes network utilisation, minimizes request blocking probability, and thus maximizes the provider's revenues. The basic idea behind our proposal is to adjust reservations, upon receipt of a new request, upon departing an existing request or upon service degradation, in a way to maximize the amount of reserved resources and minimize the number of requests rejected due to resources shortage. Our approach assumes that (a) a request includes a set of acceptable QoS values; and (b) one type of resources (e.g., bandwidth or CPU); we are currently working on extending the proposed approach to allow for two or more types of resources that are interrelated (e.g., bandwidth and CPU). It will be the subject of a future paper submission.

More specifically, we present an optimization problem formulation that optimizes the resource utilization/provider revenues and analyze its performance compared with a classical approach. The essence of our approach is to model resource allocation as an integer-programming problem and develop heuristics to solve, with an acceptable response time, the resulting optimization problem.

The remainder of the paper is organized as follows. Section 2 presents the optimization problem formulation. Section 3 describes a first heuristic, called LOH (Local

Optimization Heuristic), for the problem resolution. Section 4 presents a second heuristic, called GOH (Global Optimization Heuristic), for the problem resolution. Section 5 presents simulation results. Finally, Section 6 concludes the paper.

II. OPTIMIZATION PROBLEM FORMULATION

Integer programming is a technique for solving certain kinds of problems: maximizing the value of an objective function subject to constraints, where the objective function and constraints are all linear expressions. In the following, we describe a model based on binary integer programming technique and discuss how this model can be used to overcome the grid specific challenges, discussed in the introduction, in solving the resource reservation problem.

We formulate the proposed model as binary integer programming (IP) problem. Three inputs are required: a set of binary variables, an objective function, and a set of constraints (both the objective function and the constraints must be linear). IP attempts to maximize or minimize the value of the objective function by adjusting the values of the variables while enforcing the constraints. The resolution of the IP model consists of the optimal value of the objective function and the final values of the variables. In the following, we present how the proposed adjustment of resources reservation is mapped to an IP problem.

Let us define the following variables:

- *n*: represents the number of resources of the same type (e.g., a resource can be a CPU/server or an LSP in a MPLS network).
- m: represents the number of clients/requests currently being served in the system.
- E_i: represents the number of acceptable quality values for a request i (e.g., {10 Mbps, 5 Mbps, 1 Mbps}; in this case E_i=3).
- r_{ijk}: represents the portion of resource j used to satisfy the k^th acceptable quality value of request i.
- Rj,max: represents the maximum capacity of resource j.
- c_{ijk}: represents unit cost when resource j is used to satisfy the k^th acceptable quality value of request i.
- x_{ijk}: represents a binary variable; it assumes 1 when the amount r_{ijk} of resource j is affected to request i; otherwise, it assumes 0.
- te_i: represents the end time of the request i.
- ts_i: represents the start time of the request i; te_i-ts_i represents the duration of the request.

By means of these variables, the model can be formulated as the following integer program.

Objective function

$$\text{Max} \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{Ei} c_{ijk} * r_{ijk} * x_{ijk}(tei - tsi) \qquad (1)$$

Constraints

$$\sum_{j=1}^{n}\sum_{k=1}^{Ei} x_{ijk} = 1 \quad \text{For i=1...m (client)} \qquad (2)$$

$$\sum_{i=1}^{m}\sum_{k=1}^{Ei} r_{ijk} * x_{ijk} \le R_{j,\max} \quad \text{For j=1 ... n (resource)} \qquad (3)$$

$$x_{ijk} \in \{0, 1\} \qquad (4)$$

The objective function (1) represents the provider's revenues to be maximized. Constraint (2) ensures that a request will be supported by one resource among *n* existing resources, constraint (3) ensures that the total amount of resources allocated does not exceed the maximum capacity of each resource, and constraint (4) ensures that the variables are binary.

III. PROBLEM RESOLUTION

The goal is to have an optimal solution (i.e., optimal adjustment of resources reservation) any time changes occur in the system. More specifically, a new user request, an existing request termination, or service degradation requires the resolution of the integer program (see Section 2). The exact optimal solution of the problem can be easily computed using any Operational Research tool (e.g., Cplex [13]); however, this will incur an unacceptable response time (e.g., in terms of hours and days), especially for large number of requests and resources.

The response time is of critical concern. Indeed, the resolution process should last a very short period of time (e.g., less than a second) to be useful. Otherwise, the system will not be able to provide a response (e.g., in response to a user request) in an acceptable time. In this paper, we define two heuristics to solve the integer program in a very short period of time compared to the exact solution. In the following, we describe the first proposed heuristic, called LOH (Local Optimization Heuristic), behaviour for each event that triggers the resolution of the program. The second heuristic, called GOH (Global Optimization Heuristic), is presented in Section 4.

A. LOH: New request

In this case, the system receives a user request Di that includes a list, (Q1, Q2,...,Qm), of acceptable quality values where Q1 represents the minimum acceptable quality and Qm the most desirable quality. The system maps the list of qualities to a list of resources (RQ1, RQ2, …, RQm) that are needed to satisfy the requested qualities (e.g., to satisfy Q1, an amount of resources equal to RQ1 is needed). When dealing with a quality, such as bandwidth and CPU cycles (the case of most Grid applications), the mapping is straightforward since the quality and the corresponding amount of resources are the same.

The proposed heuristic starts by determining the resource j (e.g., computer j or LSP j) that has the most available resources $R_{\text{avail, max}}$ (e.g., the most available CPU cycles or the most available bandwidth) among the *n* resources (e.g., *n* computers or n LSPs) under consideration. Then, it checks whether $R_{avail,\ max}$ is bigger than RQ1 (i.e., $R_{avail,\ max}$>=RQ1). If the response is yes, then the heuristic determines RQj<= $R_{avail,\ max}$ (1<=j<=m), such that j=m or RQj+1> $R_{avail,\ max}$, and then allocates RQj to satisfy Qj of Di.

If $R_{avail,\ max}$ is smaller than RQ1, the heuristic determines resource j, such that $R_{availRed,\ max} = R^j_{avail} + R^j_{red} = \max(R^i_{avail} + R^i_{red})$ for 1<=i<=n where R^i_{avail} is the amount of available resources for resource i and R^i_{red} is the total amount of resources that can be reduced from requests currently being served. Let us note that the amount of resources R allocated to request k being served by resource i can be reduced by an amount equal to the difference between R and the amount of resources required to satisfy the minimum acceptable quality of request k; the amount of reduction is equal to zero if request k is minimally accommodated by the system (i.e., the system reserved just enough resources to support the minimum acceptable quality of request k). Then, the heuristic checks whether $R_{availRed,\ max}$ is bigger than RQ1 (i.e., $R_{avail,\ max}$>=RQ1). If the response is yes, then the heuristic selects a number of requests (being served by resource j) for which the amount of allocated resources is reduced (to satisfy a lower but still acceptable quality) in a way that the amount of available resources for resource j together with the sum of reductions can accommodate RQ1. In this case, the reservations of a number of active requests are updated with the new values (after reduction).

If $R_{availRed,\ max}$ is smaller than RQ1, then the request cannot be satisfied by a single resource. In this case, the heuristic determines resource m, such that $R^m_{avail} + R^m_{red} = \max(R^i_{avail} + R^i_{red})$ for $1 \le i \ne j \le n$. Then it checks whether $(R^j_{avail} + R^j_{red})+(R^m_{avail} + R^m_{red})$ is bigger than RQ1; if the response is yes, a number of requests (being served by resources j and m) for which the amount of allocated resources is reduced (to satisfy a lower but still acceptable quality) in a way to accommodate RQ1. Otherwise, the same process is repeated to consider other resources. The process terminates when RQ1 can be accommodated or when all n resources are considered without success.

B. *LOH: request termination*

A request termination (i.e., session corresponding to the request terminates) leads to resources being released. Thus, it is an opportunity to provide better quality to requests currently being served if not already getting the most desirable quality (included in the list of acceptable qualities of the request)

The heuristic determines, randomly, requests currently being served by resource j (the terminating request has been served by resource j) and increases their quality (in accordance to the list of acceptable qualities for each request) using the amount of released resources by the terminating request. With this operation, the resources utilisation is optimized and thus the provider's revenues.

C. *LOH: quality degradation*

Quality degradation occurs generally in case of failure; indeed, a partial or full failure of a resource will degrade the quality of requests currently being served by this resource. In the worst case scenario, the service provided to these requests is terminated (e.g., computer failure or LSP failure). The proposed heuristics operates in case of quality degradation in the same way as in the case of a new request. Indeed, all failed requests are processed as new requests (see Section 3.A). One of the key challenges is to minimise service disruption; this can be achieved if the response time to failure(s) is minimized.

IV. GLOBAL OPTIMIZATION HEURISTIC

The heuristic presented in Section 3 enables local adjustment of resources in order to accommodate a new request, respond to quality degradation, or respond to request termination. We believe that these adjustments provide "local" optimization and further global optimisation is possible. Indeed, if all resources and all requests being served are considered, a global optimisation is needed.

In this section, we propose a second heuristic, called Global Optimization Heuristic (GOH), which can be used in conjunction with the first heuristic (LOH). The first heuristic can be used to produce an initial solution that can be optimized, periodically for example, running GOH in the background.

GOH is based on an ejection chain neighbourhood applied to GAP (generalized assignment problem) [11, 12]. It consists of moving more than one task from the current agent to a new agent. The neighbourhood structure based on ejection chains (NSEC) was initially introduced by Glover [11] and has been applied to several problems since then. NSEC consists of two phases. In the first phase, NSEC removes a task i from an agent j, then assigns task i to a different agent w ($w \ne j$). If this does not succeed (agent w cannot accommodate task i), NSEC, in the second phase, removes a task i from an agent j, then assigns task i to a different agent but only after removing a task k from agent w and assigning task k to another agent z ($z \ne w$ but it may be equal to j). In this paper, an agent represents a resource (e.g., computer or LSP), and a task represents the amount of resources required to satisfy an acceptable quality.

With some changes to NSEC, we define our heuristic GOH as follows. First, let us define S* and L.

- Let $S^*=(X_1(x_{1.1.1}, x_{1.1.2},\ldots, x_{1.\ n.\ e}), X_2(x_{2.1.1}, x_{2.1.2},\ldots, x_{2.\ n..e}),\ldots, X_m(x_{m.1.1}, x_{m.1.2},\ldots, x_{m.\ n..e}))$ be the initial solution (e.g., given by LOH), where $Xi(x_{i.1.1}, x_{i.1.2},\ldots, x_{i.n.e})$ represents a vector of n*e binary variables associated with request Di(q1,q2,..qk,..,qe); $x_{i.j.k}=1$ if $q_{k,}$ among e acceptable qualities, is accommodated using resource j; otherwise, $x_{i.j.k}=0$. Only a single variable assumes 1 while all others are equal to 0 (see details in Section 3).

- Let ‘L’ represents the list of requests for which a quality improvement is possible. A request Di(q1,q2,..qk,..,qe) belongs to L if and only if the amount of resources reserved for Di supports qj with $j \neq e$ (assuming that qe is the most desirable quality and q1 is the minimum acceptable quality).

The operation of GOH can be summarized as follows. GOH selects, randomly (or traverses the list L starting from the first element), a request D_h that belongs to L; D_h is associated with X_h (x_{h11}, x_{h12}… $x_{h.\ n..e}$) that belongs to S*. If $x_{h.\ n..e} \neq 1$, then the following steps are executed.

- Let Rq_k (required to support q_k) represents the amount of resources in resource j reserved to D_h. GOH determines resource m, among n resources, with the most available resources R_{avail}.
- If R_{avail} >= Rq_{k+1}, then GOH selects resource m to support D_h with the best quality possible q_l; in this case, Rq_l<= R_{avail} and (Rq_{l+1}> R_{avail} or l=e)> in the worst case scenario l=k+1.
- Otherwise, GOH determines a request D from list L that satisfies the two conditions: (1) the amount of resources, Rq, currently reserved for D, in resource z, is smaller than Rq_k; and (2) Rq+available resources in z=>Rq_{k+1}. If D exists then, GOH reserves resources in resource z to accommodate the best quality possible for D_h (the minimum is q_{k+1}) and reserves resources in resource j to accommodate q for D; in this case, GOH enables better utilisation of resources by providing better quality for D_h.
- The process is repeated until all requests in L are considered.

Table 1 shows the pseudo-code of the operation of GOH.
Table 1 shows the pseudo-code of the operation of GOH.

```
Begin
For h=0 to i do
 If (X[h][2] != e and $q_{k+1}$ <= av)  /*  $q_{k+1}$ > $q_k$ */  Then
   g:=e;
   While ($q_g$ >  av) do
         g:=g-1;
   End-while;
   RV:=RV-p[(X[h][2])]* (X[h][2]);
   R[m][2]-=(D[h][g]-X[h][2]); /*availability decreased*/
   R[m][3] +=(D[h][g]-X[h][2]); /*reducibility increased*/
   D[h][5]:=m; /*   D[h]  moved to  resource m */
   X[h][1]:=g; /* index of value reserved to D[h] */
   X[h][2]:= D[h][g]; /* value reserved to D[h] */
   RV:=RV+p[(X[h][2])]*(X[h][2]);
   update A[1][2] and AR[nr][3] /* order based on
                                   available  amount;*/
   update m;/*with resource having maximum availability*
   update av;/*with amount available in m*/
 Else-If (X[h][2] != e and $q_{k+1}$  > av)  Then
     j:=0 ;  lin:=X[h][1]; /* index of quality being served to
                                          request h */.
   while (j<i)
     if (D[h][lin]>=X[j][2] and h!=j )  Then
       id=X[j][0];/*index of  resource  to where D[h]
                         moves*/
       inter=R[id][2]+X[j][2];
       If  (D[h][lin+1]<= inter)  Then
         k=4;
         while ( inter<D[h][k])
             {  k--;}
         End-while
         If (D[h][k]!=D[h][lin]) Then
           inter1=X[h][2];  inter2=D[h][k];
           id1=X[h][0];
           R[id][2]+=(X[j][2]-D[h][k]);
           R[id][3]-=((X[j][2]-D[j][1])-(D[h][k]-D[h][1]));
           R[id][4]-=((D[j][e]-X[j][2])-(D[h][e]-D[h][k]));
           R[id1][2]+=(D[h][lin]-X[j][2]);
           R[id1][3]-=( (X[h][2]- D[h][1])-(X[j][2]- D[j][1]));
           R[id1][4]-=((D[h][e]-X[h][2]) -(D[j][e]-X[j][2]));
           X[j][0]=id1;  X[h][0]=id;
           X[h][1]=k;  X[h][2]=inter2;
           RV = RV-inter1*p[inter1]+inter2*p[inter2];
         End-if
       End-if
     End-if
   End-While
 End-else-if
 End-if
End-for
End
```

Table 1. GOH: Pseudo-code

V. SIMULATIONS RESULTS

In this section, we evaluate and compare a number of schemes via simulations. More specifically, we define 4 schemes: (1) Classical scheme: It enables the support of the best quality available to a new request; however, quality remains unchanged during the session (i.e., duration of the request). If the minimum quality cannot be supported the request is rejected. This scheme corresponds to the behaviour of existing schemes; (2) LOH; (3) LOH+GOH: It represents a combination of LOH and GOH; LOH is used to give the initial solution while GOH is launched periodically in the background to perform global optimization; and (4) Exact solution of the IP problem.

In sub-section A, we compare the classical scheme, LOH, and LOH+GOH in terms of revenues and rejection ratios. In sub-section B, we evaluate LOH+GOH while in sub-section C we compare LOH+GOH against an exact solution (of the IP model described in Section 2).

We implemented the schemes using C++; we used CPLEX, running on Linux Machine, to resolve LOH and determine an exact solution. Table 2 shows the values of the simulation parameters. We use a simple price function: prix(x) = 5 times x, where x is the quality. More sophisticated price functions can be used.

Simulation parameter	Value
Number of resources:	1000
Capacity of each resource	15
Number of requests	6000
Number of acceptable qualities	4
quality	Random integer between 1 and 9
Request inter-arrival time	Random between 0 and 2 seconds
Request duration	Random between 15 and 60 minutes
GOH Optimization periodicity	90 seconds

Table 1. Simulation parameters

A. *REVENUES AND REJECTION RATIO*

Figure1 shows clearly that LOH+GOH generated the most revenues when 6000 requests were served. Classical scheme generated, in the case of 6000 requests, a revenue which is almost 4 times smaller than the revenues generated by LOH+GOH; more specifically after 6000 requests have been served, classical scheme generated 5.800E+08 while LOH+GOH generated 1.94E+09. LOH generated revenues better than classical scheme but two times smaller than LOH+GOH.

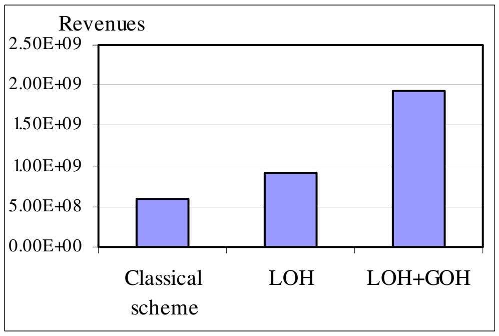

Figure1. Revenues Vs schemes

Figure 2 shows that 50% of the requests are rejected when using classical scheme; this is caused by the fact that this scheme allocates the maximum requested quality, whenever possible, without any changes (i.e., reduction) of quality during the session. Using LOH the rejection ratio is about 6%; this is explained by the fact that LOH uses CPLEX on a subset of resources. CPLEX tries to exploit, in order to accept a new request, the availability in a set of resources (in opposition to a single resource) and possibly reduces the qualities of requests being served; this definitively increases the chance of accept a new request.

While LOH+GOH supports similar rejection ratio compared to LOH, it outperforms it in terms of revenues. This can be explained by the fact that LOH+GOH uses GOH in background and this increases considerably the use of resources and thus the revenues.

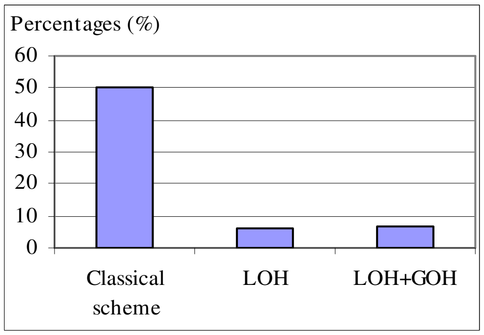

Figure 2. Rejection ratio Vs. schemes

B. *LOH+GOH: REVENUES VS. PERIODICITY*

The purpose of this section is to see how the revenues generated by LOH+GOH evolve when the periodicity of executing GOH in the background changes.

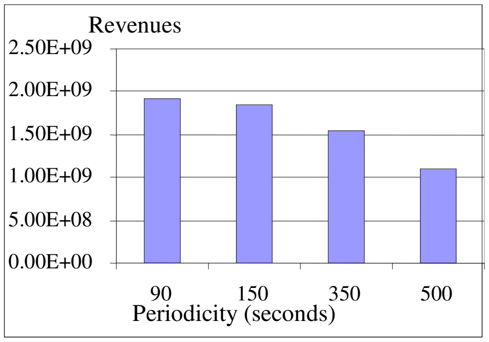

Figure 3. Revenues Vs. Periodicity

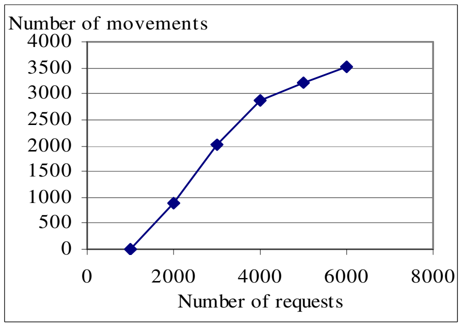

Figure 4. Movements Vs requests (using LOH+GOH)

Figure 3 shows that using smaller periodicity for running GOH in the background gives bigger revenues. However, running GOH many times in the background may cause requests be moved many times among resources; this may be not acceptable in terms of the cost of moving requests (e.g., moving a process from one machine to another machine or rerouting traffic from one LSP to another LSP). This being said, our simulation results indicate that the number of movements (between resources) per request does not exceed one movement for the duration of the request; Figure 4 shows that just over 50% of each 1000 requests are moved from one resource to another resource.

C. LOH+GOH Vs. EXACT SOLUTION

In this section, we compare LOH+GOH against an exact solution of the IP problem (see Section 2) in terms of revenues, number of movements per request, and response time. In general, getting an exact solution with realistic parameters (i.e., large size problem) is almost impossible, and in order to make this comparison successfully we have changed the values of the following simulation parameters: (1) Number of resources= 100 (instead of 1000); and (2) Number of requests= 170 (instead of 6000). We used Cplex for the exact solution.

Figure 5 shows that the exact solution generates slightly more revenues than LOH+GOH; however, the difference is minimal.

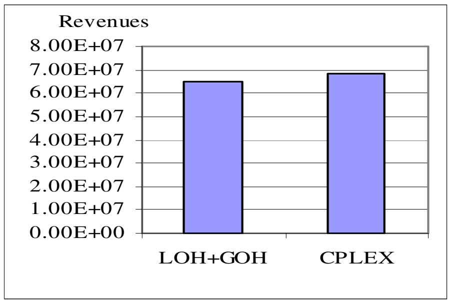

Figure 5. Revenues: LOH+GOH Vs. Exact solution

Figure 6 shows the number of movements (by the end of the simulations), per each request, executed by Cplex to produce an exact solution. The number of movements decreases when the number of requests increases. This can be explained by the fact that, for each Cplex execution processing a new request, already accommodated requests are moved trying to accommodate the new request. For example, request 7 has been moved 135 times by the end of the simulations (i.e., after the processing of request 150); this represents 135 movements in 143 (150-7) executions of Cplex; it corresponds to almost one movement per Cplex execution (135/143). This is not acceptable for most of applications. With LOH+GOH (Figure 4), a request is moved, in average, one time by the end of the simulations.

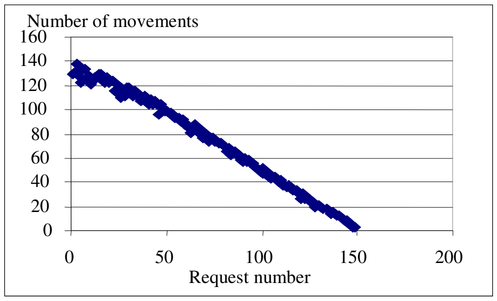

Figure 6. Exact solution: Requests Vs. movements

Figure 7 shows that the response time varies from 0 to 270 seconds (4min 30seconds) when executing Cplex to determine an exact solution while it is less than one second when using LOH. When the number of requests exceeds 170, Cplex was not able to return an exact solution after several hours of execution (not shown in Figure 7).

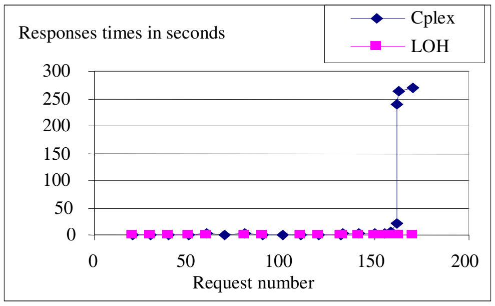

Figure 7: Responses time Vs Requests

D. ANALYSIS

We conclude that LOH+GOH is suitable to resolve the adaptive provisioning issues in Grid applications/services or any other distributed applications that require QoS assurances. In the following, we summarize our findings related to the proposed solution.

- The response time is smaller than 1 second; computing an exact solution causes a response time to exceed the 5 minutes for small size problems (see figure 7); for large size problems, the response time is "infinite".
- At a maximum a single movement, per request, between resources is required. Indeed, just over 50% of requests are moved once between resources (see figure 5). Computing an exact solution causes several movements, per request, between resources (see figure 6). The cost of these movements is not acceptable for most applications.
- It generates slightly less revenues than the exact solution. This is a small price to pay in order to produce results with acceptable response time and fewer movements of requests between resources.

VI. CONCLUSION

In this paper, we proposed an adaptive scheme that maximizes network utilisation, minimizes request blocking probability, and maximizes the provider's revenues. The basic idea behind our proposal is to adjust reservations, upon receipt of a new request, upon departing an existing request or upon service degradation, in a way to maximize the amount of reserved resources and minimize the number of requests rejected due to resources shortage.

More specifically, we developed an optimization model using BIP (binary integer programming). Then, we defined a heuristic (LOH) to resolve BIP with an acceptable (a) response time and (b) number of movements per request. A second heuristic was developed in order to be used in combination with LOH for better optimization of resources utilization. The simulation results show that LOH+GOH outperforms existing provisioning schemes in terms of revenues and rejection ratio. It also provides better performance than an exact solution while providing slightly less revenues.

Currently, we are working on extending the proposed model to consider more than one type of interrelated resources (e.g., CPU and bandwidth).

REFERENCES

[1] R. Al-Ali, A. Hafid, O. Rana and D. Walker, An Approach for QoS Adaptation in Service-Oriented Grids, The Journal of Concurrency: Practice and Experience, Vol. 16, Issue 5, Pages 401- 412 , 2004.

[2] A. Hafid and G. Bochmann, Quality of Service Adaptation in Distributed Multimedia Applications. ACM Springer-Verlag Multimedia Systems Journal, Vol. 6, No. 5, Pages 299- 315, 1998.

[3] I. Foster and A. Roy, A Quality of Service Architecture that Combines Resources Reservation and Application Adaptation. In Proceedings of 8th International Workshop on Quality of Service (IWQOS 2000), pp. 181-188, 2000.

[4] D. A. Reed and C. L. Mendes, Intelligent Monitoring for Adaptation in Grid Applications, In Proceedings of the IEEE, Vol. 93, Issue 2, Pages 426- 435, February 2005.

[5] I. Foster and C. Kesselman, eds., The Grid 2: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Francisco, 2003.

[6] R. L. Ribler, J, S. Vetter, H Simitci, and D. A. Reed, Autopilot: Adaptive Control of Distributed Applications, In Proceeding of 7th IEEE Symp. On High Performance Distributed Computing, 1998.

[7] B. Li and K. Nahrstedt, A Control-Based Middleware Framework for Quality-of-Service Adaptations, IEEE, Vol. 17 Issue 9, Pages 1632- 1650, September 1999.

[8] J.Li, R.Yahyapour, A Negotiation Model Supporting Co-Allocation for Grid Scheduling, The 7th IEEE/ACM International Conference on Grid ComputingThe 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Sept 28-29, IEEE Computer Society Press, 2006.

[9] M. Siddiqui, A. Villaz and T. Fahringer, Grid capacity planning with negotiation-based advance reservation for optimized QoS, In Proceedings of the 2006 ACM/IEEE conference on Supercomputing, No. 103, November 2006.

[10] B. Doshi, S. Wang, P. Kim; D. Goldsmith, B. Liebowitz, K. Park, Cooperative Service Level Agreement, Military Communications Conference, Pages 1-7, 23-25 Oct 2006.

[11] F. Glover, Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problem, Discrete Appl. Math., Vol 65, No.1- 3, Pages 223-253, March 1996.

[12] M. Yagiura, T. Ibaraki, F. Glover, An Ejection Chain Approach for the Generalized Assignment Problem, Technical Report #99013, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, 1999.

[13] ILOG CPLEX. http://www.ilog.com.