APPLYING HEURISTIC METHODS FOR JOB SCHEDULING IN STORAGE MARKETS

Journal:	*18th European Conference on Information Systems*
Manuscript ID:	ECIS2010-0380.R1
Submission Type:	Research-in-Progress Paper
Keyword:	Decision support systems (DSS), Algorithms, Optimization, Market engineering

APPLYING HEURISTIC METHODS FOR JOB SCHEDULING IN STORAGE MARKETS

Finkbeiner, Josef, University of Freiburg, Kollegiengebäude II, Platz der Alten Synagoge, 79085 Freiburg, Germany, josef.finkbeiner@is.uni-freiburg.de

Bodenstein, Christian, University of Freiburg, Kollegiengebäude II, Platz der Alten Synagoge, 79085 Freiburg, Germany, christian.bodenstein@is.uni-freiburg.de

Schryen, Guido, University of Freiburg, Kollegiengebäude II, Platz der Alten Synagoge, 79085 Freiburg, Germany, schryen@gmx.net

Neumann, Dirk, University of Freiburg, Kollegiengebäude II, Platz der Alten Synagoge, 79085 Freiburg, Germany, dirk.neumann@is.uni-freiburg.de

Abstract

In double-sided markets for computing resources an optimal allocation schedule among job offers and requests subject to relevant capacity constraints can be determined. With increasing storage demands and emerging storage services the question how to schedule storage jobs becomes more and more interesting. Since such scheduling problems are often in the class NP-complete an exact computation is not feasible in practice. On the other hand an approximation to the optimal solution can easily be found by means of using heuristics. The problem with this attempt is that the suggested solution may not be exactly optimal and is thus less satisfying. Considering the two above mentioned solution approaches one can clearly find a trade-off between the optimality of the solution and the efficiency to get to a solution at all. This work proposes to apply and combine heuristics in optimization to gain from both of their benefits while reducing the problematic aspects. Following this method it is assumed to get closer to the optimal solution in a shorter time compared to a full optimization.

Keywords: Decision Support System, Algorithms, Optimization, Market Engineering.

1 INTRODUCTION

Numerous web 2.0 applications have shown that there is an increasing demand for storage capacities where data has to be available and accessible via internet permanently. Online video hosting and streaming is getting more and more popular and can therefore be referred as a good example. According to comScore (2010) about 170 million US internet users watched online videos and nearly 31 billion videos were viewed from US properties during November 2009 alone. Among these the most prominent online video platform YouTube did achieve almost 40% of all US online videos being watched followed by Hulu.com with a share of only 3% and various other platforms with smaller shares. Another example for increasing storage demand in the age of web 2.0 is the social network facebook. According to their own statistics facebook is hosting more than 350 million active users, where user generated content is updated on a daily basis. Here a major storage driver is certainly uploading and sharing user photos. More than 2.5 billion photos have to be stored on the facebook platform each month (facebook, 2010). Also new technology innovations are about to benefit from storage space which has to be available not only locally but also on a global scope. On the occasion of the Consumer Electronic Show 2010 for example, the computer hardware manufacturer ASUS has showcased future technology concepts, presenting both mobile and stationary devices always connected with the cloud. (see ASUS, 2010)

Besides the growing storage requirements from web 2.0 and future web 3.0 applications, there are also economic reasons for an increasing online storage demand: Carr (2005) has shown that 50% to 60% of the data storage of corporately operated data centers is idle. With the emerge of storage clouds and storage-as-a-service offerings it is now possible to size-down these data centers and buy online storage whenever it is required by peak loads. Data center operation costs can therefore be cut down significantly.

Another interesting observation has been done by Broberg et al. (2009): Due to expensive service costs, content delivery networks (CDNs) like Akamai or MirrorImage were only attractive to large enterprises in past times. Being able to resort to storage cloud providers such content delivery network services are now also practicable for smaller size companies as shown with MetaCDN.

Computing resources can be offered on centrally managed markets, where the scheduling of resource requests and offers becomes a major topic for information systems research. The crucial winner determination problem, i.e. which requester obtains what resource at what time, can be handled using the exact run of an optimization program. Such optimization runs tend to be less useful in practice since they simply take too long, where market characteristics require high scaled order sizes to be scheduled in seconds. Heuristic methods can overcome the computational complexity of the exact optimization problem by determining an approximation to the optimal solution in polynomial time. This approach can be seen as less satisfying though, since it constitutes only a suboptimal solution, save when used in conjunction with the optimal solver.

Modern modeling languages and solvers allow the feeding of initial values into an optimization program to improve a starting solution towards a better solution. Grabbing this idea and applying it in the domain of scheduling problems, the contribution of this paper is to introduce heuristic optimization – a method to use good heuristic solutions as initial value in an exact optimization run. The computation performance (i.e. runtime and iterations) is assumed to be better with heuristic optimization than with convenient optimization runs without providing starting values. In the line of explanations an allocation algorithm that has to fit for a double-sided storage market is set up, a corresponding greedy allocation scheme is applied which serves as basis for building up the heuristic optimization method.

This work is structured as follows. In section 2 related work in the field of heuristic methods and storage markets is presented. In Section 3 the model framework is developed briefly and the heuristic methods are explained in detail. A storage allocation problem serves as basis for the model. A

heuristic is set up in order to deliver initial values which are required for the heuristic optimization method being explained gradually at the end of section 3. In the subsequent section 4, first simulations are run for chosen market sizes. Initial results are critically analyzed and compared. Finally, section 5 summarizes and concludes the paper. Besides, further research activities are addressed in an outlook.

2 RELATED WORK

2.1 Heuristic Methods

Approximation techniques are commonly used in operations research to get to a good solution when being confronted with difficult problems (i.e. NP-complete or NP-hard). Such techniques are classified as heuristic methods. With these approaches optimality cannot be assured. (Winston and Venkataramanan, 2003)

Zanakis et al. (1989) developed a scheme that categorizes 442 research articles with respect to the application area and the type of heuristic method used. They have found that the most frequently used methods for scheduling problems are in the class 'construction' followed by 'improvement' algorithms.

Construction methods – typically a greedy approach is used – aim to yield a good solution and often focus on providing validity with the solution that has to be reached. Improvement methods take a given valid solution looking (iteratively) for a better one.

For the underlying job scheduling problem in this paper, we stick to these two types of heuristic methods, since they are easy to develop and implement. First a simple greedy heuristic will be used to get a good feasible solution (i.e. construction method). Then the greedy solution will serve as starting value for the heuristic optimization method (i.e. improvement method).

The basic form of the greedy heuristic being used in this work can be found in Lehmann et al. (2002). Stößer et al. (2010) proposed to adjust the basic greedy allocation scheme to fit for 'double-sided multi-attribute auctions with timeslots'.

2.2 Storage Services

Hasan et al. (2005) analyzed the evolution of storage service providers (SSP) from both perspectives - technical and business. Economic incentives but also challanges of outsourcing storage are worked out briefly and in addition two case studies are presented.

Placek and Buyya (2006) determined the relevant attributes for trading storage and setting up storage policies to be storage capacity, upload rate, download rate and time frame. For specifying storage job requests and offers, we take this set of attributes skipping upload and download rates and replacing them by the term 'data transfer' since storage services usually do not only price the capacity which is used during a storage job but also the overall amount of data that is transferred while a job proceeds.[1]

3 THE MODEL FRAMEWORK

This section elaborates a market-based storage scheduling model which allows agents to place their bids in order to buy storage capacities and nodes to supply storage space at which an automated allocation mechanism decides if and which storage request is optimally handled by which node. Besides an associated heuristic counterpart is presented and together with the optimization model it forms the basis for examining the performance of optimization tasks when the heuristic solution is set as the initial value for running the optimization.

[1] For example: Pricing of Amazon Simple Storage Service (Amazon S3). Available at http://aws.amazon.com/s3/#pricing

The next part presents the general setting of the model. Section 3.2 shows the mathematical representation of the storage optimization model and 3.3 presents a greedy allocation scheme of the model. This section is concluded with subsection 3.4 which introduces and explains the heuristic optimization method.

3.1 The General Setting

There are two parties in the market where storage space is the central scarce resource to be traded: Requesters who seek to maximize their private utilities by obtaining storage capacities and on the other hand providers (i.e. datacenter managers) that are willing to maximize their earnings by supplying and selling storage services. A node N can offer storage services up to its total storage capacity limit. Furthermore it is assumed to have a data transfer limit which restricts the acceptance of storage jobs. A Job J is essentially characterized by a storage request which has to be executed over a certain time horizon. Providers and job requesters meet at the storage market which is assumed to be centrally managed.

All market participants are able to submit their node offerings and job requests to a market mechanism which collects them for a certain time period. After this bidding phase a scheduler allocates the offers and requests according to a specific algorithm. This market mechanism can be compared to a bulletin board where offers and requests are collected first and after this a scheduler is responsible for allocating and coordinating the jobs to the storage servers. In accordance with Parkes et al. (2001) we propose to use a sealed-bid mechanism which states that the market participants do not get to know the requests or offers of the others. The scheduler is assumed to have perfect information being able to determine an optimal allocation schedule. After successful allocation the market process can be repeated. This is initiated by beginning a new bid collecting phase.

A provider has to offer the storage services of one of its nodes in the following way: (r_n, s_n, d_n, f_n, l_n). In order to run the node a reservation price r_n must be paid at least to the provider. s_n denotes the maximum storage space which is available on a node and its total data transfer capability is specified by d_n. To cover the case that a node is not available during the whole time horizon under consideration storage providers must also provide information about the first period f_n a node is available as well as the last period l_n before the node is shut down. A requester who wants to obtain storage space must submit the following job bundle (v_j, s_j, d_j, f_j, l_j) to the market. Besides their storage demand s_j and their valuation v_j which is expressed in monetary units per storage unit requesters must also provide information about the data transfer d_j which is needed for fulfilling a job. All of these characteristics are valid for each time slot. Finally the requester has to state her time preferences – i.e. first f_j and last l_j time slot the storage request has to be performed.

Storage jobs can only be performed if enough storage and data transfer capacity is left in all required time periods on the assigned node. At this point it is important to emphasize that a node can handle multiple storage jobs at the same time as long as the storage and data transfer capacity limitations as well as the time restrictions are not exceeded. On the other hand it is not allowed to separate a job with respect to its time or storage space dimension to let it run on multiple nodes. The job is assumed to be inseparable and thus it can only be performed in its entirety on one single node.

Node n	r_n	s_n	d_n	f_n	l_n	*Job j*	v_j	s_j	d_j	f_j	l_j
n1	1	137	193	1	8	*j1*	15	83	68	3	7
n2	3	155	138	2	7	*j2*	1	52	155	5	10
n3	6	137	157	1	9	*j3*	3	61	86	2	8
n4	2	74	125	1	10	*j4*	19	59	81	1	3
						j5	18	80	118	2	5
						j6	14	58	108	3	6
						j7	3	95	115	1	5
						j8	8	28	109	2	7

Table 1. Sample node offers and job requests

Example: Table 1 shows an exemplary market board of the storage market under consideration. Node *n1* for example offers 137 storage space units and a total of 193 units of data transfer for each period in the time span from timeslot 1 to 8. Job *j1* demands for 83 storage units and 68 units of data transfer in the time horizon from timeslot 3 to 7.

3.2 Mathematical Representation

This subsection presents the exact algorithm to determine the optimal allocation in the storage market. The allocation is characterized by the binary decision variable x_{jnt}, where $x_{jnt} = 1$ if job j is allocated to node n in phase t and $x_{jnt} = 0$ if not. The set of all phases for the allocation problem is defined as the time horizon $T = \{t \in \mathbb{N} \,|f_j \leq t \leq l_j\} \cup \{t \in \mathbb{N} \,|f_n \leq t \leq l_n\}$. With this definition jobs and nodes lying out of the time horizon will not be allocated and thus they can be neglected. Defining J as the set of all job requests (storage demand) and N as the set of all defined storage nodes (storage supply) the winner determination problem can be mathematically represented in the following way:

$$\max_x W = \sum_j^J \sum_n^N \sum_t^T s_j x_{jnt} [v_j - r_n] \qquad (O1)$$

Subject to:

$$f_j \leq t \leq l_j, f_n \leq t \leq l_n, v_j \geq r_n, \qquad \forall j \in J, n \in N \qquad (C1)$$

$$\sum_n^N x_{jnt} \leq 1, \quad x_{jnt} \in \{0,1\}, \qquad \forall j \in J, t \in T \qquad (C2)$$

$$\sum_j^J x_{jnt} s_j \leq s_n, \qquad \forall n \in N, t \in T \qquad (C3)$$

$$\sum_j^J x_{jnt} d_j \leq d_n, \qquad \forall n \in N, t \in T \qquad (C4)$$

$$\sum_{u=f_j}^{l_j} \sum_n^N x_{jnu} = (l_j - f_j + 1) \sum_n^N x_{jnt}, \qquad \forall j \in J, t \in T \qquad (C5)$$

The objective (*O1*) represents an integer program of the allocation problem which has to determine an optimal allocation schedule in order to maximize the total welfare W. Assuming quasi-linear utility functions of the requester and providers, welfare can be seen as the sum of the difference of the requesters' job valuations and the providers' reserve price. So the goal is to assign the cheapest possible node to the most valuable job.

Constraint (*C1*) allows for an allocation only if it is feasible with respect to job-time accessibility ($f_j \leq t \leq l_j$) and resource availability ($f_n \leq t \leq l_n$). Hence, a job can only be allocated if its time span corresponds with the time horizon of the node. The profitability statement ($v_j \geq r_n$) ensures that the requester's willingness to pay is not being exceeded by the reserve price which would lead to a negative contribution. (*C2*) states that a job can maximally be handled by one node at the same time. (*C3*) and (*C4*) specify the resource constraints. The storage and data transfer capacities of a certain node cannot be exceeded by the respective capacity requirements of all of its assigned jobs together. Constraint (*C6*) defines that a job can only be allocated if it is fully executed in all requested time slots and rejected if this is not possible.

This allocation problem is a special case of the Multiple Knapsack Problem (MKP). Chekuri and Khanna (2006) argue that MKP is NP-complete. Since the problem at hand is more complex than

MKP solving it to optimality by use of a complete enumeration is clearly also NP-complete, thus computationally intractable in practice where users and market characteristics prefer to obtain a solution in relatively short time periods. An often applied approach to deal with the above mentioned computational hardness is to use heuristic concepts in order to get a solution near optimality in short time periods and with considerably less computation efforts. A heuristic counterpart of this allocation problem is presented in the next subsection.

3.3 Greedy Heuristic

In order to obtain a good approximation in terms of the optimal objective function value in a timely manner and especially with respect to the constraint set (*C1*) – (*C5*) a feasible first allocation scheme which subsequently will serve as initial value for running a complete enumeration of the integer program, a simple greedy heuristic is set up with the following rules.

Policy: *Allocate the job with the highest valuation to the node offering its services at the lowest reserve price.*
Allocation:

1. Sort jobs $j \in J$ in descending order of their valuations and nodes $n \in N$ in ascending order of their reserve prices.
2. Start with the most valuable job j and allocate it to the node n with the lowest reserve price at which an allocation is feasible subject to the technical constraints (*C1*) – (*C5*).
3. Repeat step 2 with the next highest $j \in (J-1)$ until there is no more job left to allocate.

Box 1: Three steps for performing the heuristic greedy allocation.

The approach of this heuristic is to maximize the difference between the valuation of a job and the reserve price [i.e. $v_j - r_n$ in objective function (*O1*)] of a node per unit of storage and time for each job allocation by sorting job requests and node offers as indicated in box 1.

Example: After sorting the exemplary market board from above as described in the heuristic rules the optimal allocation can be shown as in table 2 presented. The heuristic proposes to implement five jobs out of seven. Jobs j2, j3 and j7 are not performed at all, since there are no more resource capacities left to cover these low valuation jobs. The heuristic yields an objective value of $W^{heu} = 15{,}745$ where the exact optimal solution performs with $W^{opt} = 16{,}871$. In this example the heuristic implements the same jobs as the exact optimization but the assignment to the nodes is handled differently which yields a good but still suboptimal objective value for the heuristic solution.

X	*j1*	*j2*	*j3*	*j4*	*j5*	*j6*	*j7*	*j8*
n1	0	0	0	1	0	1	0	0
n2	0	0	0	0	1	0	0	0
n3	1	0	0	0	0	0	0	0
n4	0	0	0	0	0	0	0	1

Table 2: Heuristic allocation schedule

3.4 Heuristic Optimization

This section presents the combined approach of using heuristic solutions as starting values in the optimization run. An initially derived heuristic allocation schedule can be given over to the integer program as initial value. From this starting point the solver is assumed to improve the given suboptimal result towards a better optimal solution. The performance of this attempt can be compared to a benchmark optimization where the same market board is being optimized without using initial

values. The following steps explain the procedure which has to be accomplished in order to run a heuristic supported optimization in detail.

For a given market board the following steps have to be performed:

1. Use a modeling language in order to set up the specific allocation problem. Optional: For comparability purposes a benchmark optimization without the existence of initial values can be executed.
2. Apply the heuristic to the given market board as shown in box 1. At this step it is important to ensure that the decision variables can be delivered in a proper format to be handled by the modeling system and solver.
3. Insert the heuristic allocation schedule as initial scenario to the previously created model.
4. Run the program and optionally compare the results with the benchmark and heuristic solutions.

Example: At first a benchmark optimization upon the above mentioned market board is computed. The heuristic decision schedule presented in table 2 is implemented as initial allocation to the integer program. The optimization task is being executed anew. For this small scale problem the heuristic supported optimization improves the initial value eventually yielding the same optimal allocation schedule and objective value as the benchmark optimization does. The number of iterations is the same for both optimization tasks. Since the example covers only a small market the benchmark as well as the heuristic optimization can be computed within seconds. Comparing the time requirements of both optimization tasks however reveals an interesting result: The heuristic optimization requires only about 60% of the time which is needed in order to calculate the benchmark optimization. Projected to greater sized markets this can lead to huge computation time differences between the two optimization types.

These first results encourage to intensify research efforts in the direction of heuristic optimization methods. It is particularly interesting to see how computation time and iterations perform with larger sized problems. Initial results for more diverse market sizes are presented in the following section.

4 SIMULATION

This section presents initial results of first simulation rounds running GAMS/CPLEX on an Intel Xeon 5335 Processor with 2 GHz and 2 GB of memory.

4.1 Data creation

Different scenarios with varying market sizes are simulated where the market size of 10 indicates that 10 jobs (nodes) are requested (offered) on the market. The resource capacities are drawn from a normal distribution whereas the valuations are uniformly distributed. In order to keep the scenarios comparable the same distribution characteristics apply in every scenario. Furthermore only positive values are allowed. Table 3 shows the chosen data generation parameters at a glance.

Parameter	Resource Requesters	Resource Providers
Market size	10, 20, 50, 100, 200	
Storage space	Normal(75,15)	Normal(100,15)
Data transfer	Normal(90,20)	Normal(140,20)
Valuation	Uniform(1,20)	-
Reservation price	-	Uniform(1,13)

Table 3: Parameters for data generation

Without the loss of generality the first simulations of the allocation problem are solved for a discrete single period. Hence, for the sake of simplicity the simulated markets are optimized with a time-horizon-reduced algorithm. Since there are no time or iteration restrictions relied on the CPLEX solver

it can run until an optimal solution is being found. In the following section the performance of the heuristic optimization is compared to the benchmark optimization.

4.2 Initial results

Subsequently first simulation results are presented comparing the performance of the heuristic optimization approach to a convenient benchmark simulation run.

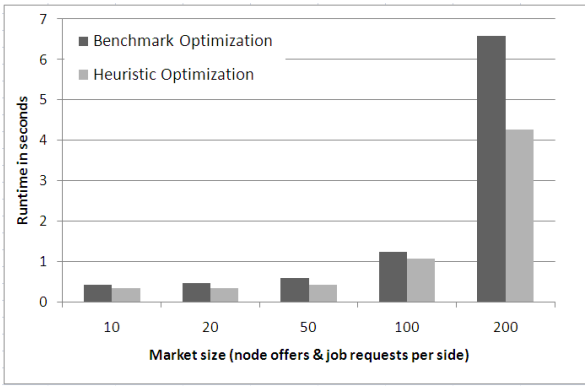

Figure 1: Runtime Evaluation

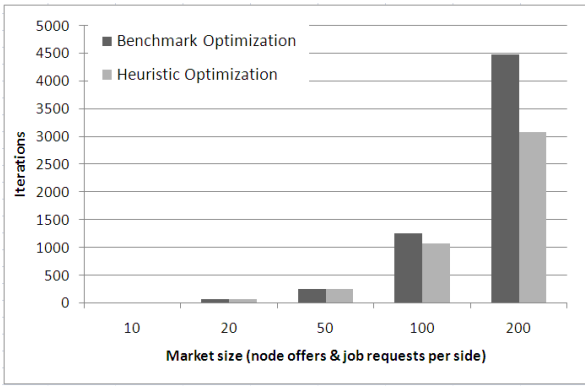

Figure 2: CPLEX Iterations

Figure 1 shows the runtime of both approaches for each market size. Considering only the benchmark optimization and comparing the runtime for market sizes 100 and 200, the hardness in computing the exact model can clearly be seen. While only about one second is needed in order to complete the optimization run for 100 offers and requests, the same procedure needs GAMS/CPLEX to take more than 6 times longer for a double market size. A set of 1000 nodes and jobs takes already about 30 minutes of runtime, which is 1800 times higher than the optimization in a market with 100 offers and requests. More interestingly the comparison of benchmark to heuristic optimization reveals new insights and first speculations: The heuristic optimization performs better in each scenario. For lower scaled scenarios (market size 100 and below) the absolute difference seems not to be so great. Comparing the relative performance of the heuristic optimization to the benchmark for these small problems it can be shown that the heuristic performs 26% faster on average than the benchmark does. For greater market boards (200 and above) the difference seems to be more striking. The heuristic optimization requires only about 69% of the runtime which is required in order to run its benchmark counterpart. For an order size of 1000 nodes and jobs the relative difference is even getting greater. While the benchmark requires about 30 minutes of runtime the heuristic optimization does only need 9 minutes.

Figure 2 shows a similar picture for the iterations which are required by CPLEX to complete an optimization run. For scenarios with market size 10 and 20 the iterations are negligibly small whereas the iteration requirement is getting pushed significantly with increasing market size, which indicates again the computational complexity of the exact algorithm. Surprisingly considering the market size 50 for both optimization runs the same 251 iterations are required. A first answer to this exception is that CPLEX seems to optimize market boards of a lower size with a complete enumeration, not only with the market size of 50 but also with a market size of 10 and 20. Considering the order size of 100 the benchmark needs 1268 iterations while with the heuristic optimization approach 1084 iterations are sufficient. Again, considering 200 nodes and jobs the difference is getting significantly great with 4477 iterations for the benchmark compared to 3078 iterations for the heuristic optimization. With 1000 order sets the solver already needs 68557 iterations for the benchmark and about 53092 iterations for its heuristic optimization counterpart.

Market Size	Heuristic Objective (Initial Solution)	Heuristic Optimized Objective	Optimal Benchmark Objective
10	2209	3785	3785
20	10568	11004	11078
50	22639	23844	23932
100	45125	45961	46516
200	87515	90024	91984

Table 4: Comparison of the average objective function values

Another interesting observation can be done by comparing the averaged objective function values of the initial solution and the optimal objective value of the heuristic optimization to the benchmark values, as shown in table 4. The heuristic optimization model indicates three cases of improving the initial values of a heuristic optimization model:

- A suboptimal initial solution is improved until the global optimum (benchmark objective) is reached. (Compare with market size 10 in table 4)
- A suboptimal initial solution is improved until a local optimum (heuristic optimized objective) is reached. The local maximum is closer to the global one but remains still suboptimal. (Indicated by market sizes 20, 50, 100 and 200 in table 4)

Besides there is another case possible for single simulation scenarios:

- A suboptimal initial solution is not being improved. The program output contains the same schedule as its heuristic initial allocation schedule.

While the first two cases as well as the time and iteration performance of the heuristic optimization approach encourage further research activities in this direction the third case can be seen as rather deflating. However, with this result different research questions (dealing with the reasons for this situation and if and how to handle it) can be brought up.

5 CONCLUSION & OUTLOOK

This research-in-progress paper has presented heuristic optimization – the idea of including heuristic solutions as initial value into a full optimization program where a storage allocation problem was built up in order to serve as basis for this approach. Initial values have been created by using an adequate greedy heuristic and delivered to the integer program to be optimized. First simulations have shown that the idea can contribute the optimization procedure to be done in a shorter time with less iterations than the benchmark optimization requires. We would like to support this result by extending the idea as well as the model in future research papers proposing to consider the following ideas and research questions.

The evaluation of heuristic optimization approaches can be enhanced and the model per se extended in several ways. First it seems obvious to intensify simulation efforts for the single period model. Special attention can be relied on running multiple simulation rounds for each order set with a greater set of market scenarios and different job / node competition ratios. A next step is to examine the performance of the heuristic optimization attempt when the time horizon is incorporated which will clearly increase the computational hardness.

A second direction for obtaining new insights is to modify the model at hand. Goal programming approaches can be implemented in order to optimize for different objectives within the same program. Such additional objectives can deal with the minimization of operating costs and especially energy costs which are seen as a major cost driver for operating datacenters. Moreover the presented approach can be applied to and evaluated with other scheduling or more generally optimization problems. Experiments can contribute to explain how the approach performs with more complex types of allocation problems.

Another question arises by considering other heuristics. The idea is to run simulations with changing heuristics as initial solution and compare the performance of these. Besides the greedy heuristic there are other heuristics with different properties available for testing the heuristic optimization approach. It is from particular interest to work out if the heuristic under consideration inherits its properties to the optimization procedure. It is aimed for classifying different heuristics with respect to their behaviour within the heuristic optimization model.

As shortly indicated at the end of the previous section it may also be analyzed in which ways the initial allocation is improved and more interestingly worked out the reasons for the potential inertia situation of the heuristic starting values which are not improved by the solver within the heuristic optimization run.

This paper is concluded with a summary of the first interim results and contributions:

- Using heuristic determined allocation schedules as initial value in an optimization run can lead to considerable time and iteration savings especially for larger scale problems compared to an optimization task without providing initial values.
- The performance of the heuristic optimization seems to depend heavily on the quality of the initial allocation. With bad starting values which are far away from global optimality the performance results for certain simulation scenarios seem to be comparable with a benchmark optimization.
- Mainly with larger scale problems, for certain scenarios the heuristic starting values may seemingly constitute a local optimum which is indicated by the solver getting stuck without improving the initial solution.
- The case of improving the initial values towards a suboptimal local optimum (not global) is also possible.

All of these results are subject to further research efforts in order to be able to provide a more general conclusion.

References

ASUS (2010). Advanced Projects: Showcasing the Technologies of Tomorrow, Today! http://www.asus.com/News.aspx?N_ID=ZKwRY8Vdou3yhQuG

Broberg, J., Buyya, R., and Tari, Z. (2009). MetaCDN: Harnessing 'Storage Clouds' for high performance content delivery. Journal of Network and Computer Applications, 32, pp. 1012-1022.

Carr, N. (2005). The End of Corporate Computing. MIT Sloan Management Review, 46 (3), 67.

Chekuri, C. and Khanna, S. (2006). A PTAS for the Multiple Knapsack Problem. SIAM Journal on Computing, 35 (3), 713-728.

comScore (2010). November Sees Number of U.S. Videos Viewed Online Surpass 30 Billion for First Time on Record. http://www.comscore.com/Press_Events/Press_Releases/2010/1/November_Sees_Number_of_U.S._Videos_Viewed_Online_Surpass_30_Billion_for_First_Time_on_Record.

Facebook (2010). Facebook Statistics. http://www.facebook.com/press/info.php?statistics

Hasan, R., Yurcik, W., and Myagmar, S. (2005). The evolution of storage service providers: techniques and challenges to outsourcing storage. In *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*.

Lehmann, D., O'Callaghan, L., and Shoham, Y. (2002). Truth revelation in approximately efficient combinatorial auctions. Journal of the ACM, 49(5), pp. 577-602.

Parkes D.C., Kalagnanam J., and Eso M. (2001). Achieving Budget-Balance with Vickrey-Based Payment Schemes in Exchanges. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence, pp. 1161-1168.

Placek M, and Buyya R (2006) Storage exchange: A global trading platform for storage services. In: Proceedings of the 12th international European parallel computing conference (EuroPar 2006). Springer, Berlin.

Stößer, J., Neumann, D., and Weinhardt, C. (2010). Market-Based Pricing in Grids: On Strategic Manipulation and Computational Cost. European Journal of Operational Research, 203, pp. 464-475.

Winston, W.L., and Venkataramanan, M. (2003). Introduction to Mathematical Programming, 4th Edition, Chapter 14

Zanakis, S.H., Evans, J.R., and Vazacopoulos, A.A. (1989). Heuristic methods and applications: A categorized survey. European Journal of Operational Research, 43, pp. 88-110.