# Heuristics for Negotiation Schedules in Multi-plan Optimization

Bo An
Dept. of Computer Science
University of Massachusetts, Amherst
Amherst, MA USA
ban@cs.umass.edu

Fred Douglis
IBM T.J. Watson Research Center
Hawthorne, NY USA
fdouglis@us.ibm.com

Fan Ye
IBM T.J. Watson Research Center
Hawthorne, NY USA
fanye@us.ibm.com

## ABSTRACT

In cooperating systems such as grids [4] and collaborative streaming analysis [2], autonomous sites can establish "agreements" to arrange access to remote resources for a period of time [1]. The determination of which resources to reserve to accomplish a task need not be known *a priori*, because there exist multiple plans for accomplishing the same task and they may require access to different resources [3]. While these plans can be functionally equivalent, they may have different performance/cost tradeoffs and may use a variety of resources, both local and belonging to other sites. The negotiation schedule, i.e., the order in which remote resources are negotiated, determines how quickly one plan can be selected and deployed; it also decides the utility for running the plan. This paper studies the problem of optimizing negotiation schedules in cooperative systems with multiple plans. We first provide a voting-based heuristic that reduces the complexity $\mathcal{O}(n!)$ of the exhaustive search to $\mathcal{O}(mn^q)$. We also present a weight-based heuristic that further reduces the complexity to $\mathcal{O}(mn)$. Experimental results show that, on average, *1)* the voting-based approach achieved 6% higher utility than the weight-based approach but the voting-based approach has a much higher computation cost than the weight-based approach, *2)* the two proposed approaches achieved almost 50% higher utility than a randomized approach; and *3)* the average utility produced by the two proposed approaches are within almost 90% of that of the optimal results with reasonable plan sizes.

## Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence** ]: Problem Solving, Control Methods, and Search

## General Terms

Algorithms, Design

## Keywords

Optimization, heuristics, collaborative systems

## 1. INTRODUCTION

In cooperating systems such as grids [4] and collaborative streaming analysis [2], autonomous sites can establish "agreements" to arrange access to remote resources for a period of time [1]. The determination of which resources to reserve to accomplish a task need not be known *a priori*, because multiple plans for accomplishing the same task are available and they may require access to different resources [3]. Given a processing task, multiple functionally equivalent plans can be generated, e.g., through a planner [9]. Each plan requires a set of resources, possibly from other sites. The site can use any candidate plan to accomplish the processing task, with different performance/cost tradeoffs. To deploy a plan, the site needs to obtain all the resources required by that plan. It makes agreements with other sites, which control access to their resources.

We have built a Negotiation Management (NM) system, which is a subcomponent of a collaborative stream processing environment [2]. This system conducts automated negotiation among multiple sites to select the best candidate sites with which to negotiate and then select the best execution plan given the obtained agreements. NM has two main modules: the *Scheduler* and the *Proposal Generator (PG)*. The scheduler receives plans from the planner and determines the negotiation schedule, i.e., the order in which resources are negotiated. Then it submits negotiation tasks to the *PG* in that order. Each negotiation task asks for one resource that appears in the plans. *PG* conducts sequential negotiations to get the resources specified in negotiation tasks. To be successful, the scheduler needs to satisfy all the resources required by at least one plan within the cost and time constraints specified for the plan.

This paper studies the optimization problem of the scheduler, i.e., finding the best negotiation schedule. The schedule determines whether any plan can be satisfied; if so, which plan is satisfiable, and at what cost. Determining the optimal negotiation schedule is non-trivial given multiple plans from which to choose and multiple resources in each plan. The scheduler also faces both *time* and *budget* constraints within which the negotiation must finish. Moreover, the outcome of negotiated resources affects the schedule of remaining resources, adding to the complexity of the problem. Naive methods such as exhaustive search and randomized approach are either too expensive ($\mathcal{O}(n!)$ complexity) or yield low performance. As alternatives, we propose heuristics that trade between complexity and utility. We first provide a voting-based heuristic that reduces the complexity to $\mathcal{O}(mn^q)$. We then present a weight-based heuristic which has even less complexity $\mathcal{O}(mn)$, with comparable utility. Experimental results show that the proposed approaches achieve almost 50% higher negotiation utility than a randomized approach and produce utilities within 90% of that of the optimal schedule with reasonable plan sizes.

The remainder of this paper is organized as follows. Section 2 summarizes related work. The optimization problem is introduced in Section 3. Section 4 introduces the voting-based heuristic and Section 5 presents the weight-based heuristic. In Section 6, we

**Cite as:** Heuristics for Negotiation Schedules in Multi-plan Optimization, Bo An, Fred Douglis and Fan Ye, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.),May,12-16.,2008,Estoril,Portugal,pp.551-558.

examine the performance of our approaches through simulation. Section 7 concludes the paper and outlines future work.

## 2. RELATED WORK

Pegasus [3] is a workflow mapping engine which can automatically map high-level workflow descriptions onto distributed infrastructures. Pegasus enables users to represent the workflows at an abstract level without needing to worry about the particulars of the target execution systems. It provides robustness and reliability through dynamic workflow remapping and automatically manages data generated during workflow execution and capturing their provenance information.

In System S [2], a job is an execution unit that accomplishes certain work through stream analysis. A job takes the form of a processing graph, consisting of resources, i.e., data sources and processing elements (PEs), which are interconnected in a certain manner. These resources might be located at multiple different sites. Due to the potentially large numbers of data sources and PEs needed in complex jobs, and the existence of functionally equivalent processing graphs, it is infeasible for human users to manually construct and identify the best alternative graph. System S has a planner component that can construct processing graphs automatically from high-level descriptions of desired results [9].

Many resources needed in plans are accessed exclusively. In order for a site to reserve a limited resource from another site, it must establish an agreement with the other site, specifying the price of sharing the resource [1]. Therefore, some mechanisms should be used by sites to dynamically establish such agreements. One example of this is automated negotiation [6] among software agents; research on engineering e-negotiation agents has received a great deal of attention in recent years (e.g., [5]).

The optimization problem in this paper can be treated as a stochastic scheduling problem. Stochastic scheduling is concerned with scheduling problems in which the processing times of tasks/jobs are modeled as random variables. Thus a job's processing time is not known until it is complete. Stochastic scheduling has been an active research area for a long time (see [8] for a survey) and a number of techniques like approximation [7] have been proposed. Our optimization problem is different from most stochastic scheduling problems in that there are multiple candidate plans for a task/job. In addition, the scheduler faces both deadline and budget constraints.

As the scheduler has multiple plans from which to choose and each plan has multiple resource requirements, the scheduler needs to determine the order in which to negotiate resources. We do not address the problem of how *PG* makes agreements in this paper.

## 3. SCHEDULE OPTIMIZATION

This section presents the optimization problem of the scheduler. Some variables used in this formulation are summarized in Table 1.

The planner generates a set of plans $\mathcal{PS}$, and each plan consists of a set of resources. The resources in plan $PL$ are $\mathcal{PL}$. Without loss of generality, we assume a resource appears at most once in any single plan.[1] The planner will then submit the plans to the scheduler and ask the scheduler to make agreements to satisfy the resource requirements of any one of the plans. The planner will also specify 1) the *reserve price* $\delta$, i.e., the highest amount of money the scheduler can use to make agreements, and 2) the *deadline* $\tau$, i.e., the longest amount of time the scheduler can use to do scheduling and ask *PG* to make agreements. The planner submits all plans to the scheduler at the same time before scheduling.

[1]Our approach can be extended to accommodate the situation that one resource appears more than once in a plan.

**Table 1: Symbols used in this paper**

| | |
|---|---|
| $\tau$ | scheduling deadline |
| $\delta$ | the reserve price for the scheduling |
| $\mathcal{PS}$ | plans submitted to the scheduler |
| $\mathcal{PS}^t$ | feasible plans at the scheduling round $t$ |
| $\mathcal{PL}$ | the set of resources in plan $PL$ |
| $t(r)$ | the negotiation time for getting resource $r$ |
| $c(r)$ | the price for getting resource $r$ |
| $p(r)$ | the probability that *PG* can get resource $r$ successfully |
| $T^t$ | the resource in the negotiation task at round $t$ |
| $\mathcal{AG}^t$ | the set of agreements at round $t$ |

After receiving plan(s) from the planner, the scheduler submits one task (a resource) to *PG*. After receiving the negotiation result from *PG*, scheduling enters the next round: the scheduler selects another task to submit to *PG*. The result of a negotiation task could be "success," which indicates that *PG* made an agreement, or "failure," which indicates that *PG* failed to get the resource.

Resource consumers can revoke agreements made before at the cost of paying a penalty. The penalty depends on the price of the agreement and the period from the agreement making time to the revocation time. In general, the higher the price, or the later the revocation of an agreement, the greater the penalty. Assume the price of an agreement $Ag$ is $Prc(Ag)$, the agreement making time is $T_{mk}(Ag)$ and the agreement revoking time is $T_{rvk}(Ag)$. We assume a linear penalty function as a function of price and duration: specifically, the penalty is $\rho(Prc(Ag), T_{rvk}(Ag) - T_{mk}(Ag)) = \min\Big(\alpha Prc(Ag)\big(T_{rvk}(Ag) - T_{mk}(Ag)\big), Prc(Ag)\Big)$, where $\alpha \in (0, 1]$. Other forms of penalty functions are possible but we take this simple form in the paper.

The planner gives the reserve price (budget) to the scheduler, whose goal is to minimize the spending for getting all the agreements for one plan. The scheduler will report the scheduling result to the planner before the deadline approaches. A scheduling result could be either a "failure" notification or a plan with a set of agreements which can satisfy the plan's resource requirements. The planner's cost includes the prices of the agreements for that plan and penalties incurred by revocation. For ease of analysis, we treat the planner's utility as the scheduler's utility. The scheduler tries its best to minimize the spending, thus maximizing its own utility, defined as

$$u = \begin{cases} (1+\beta)\delta - Prc(\mathcal{AG}) - \rho(\mathcal{AG}) & \text{if scheduling succeeds} \\ -\rho(\mathcal{AG}) & \text{otherwise} \end{cases}$$

where $Prc(\mathcal{AG})$ is the price of all non-revoked agreements and $\rho(\mathcal{AG})$ is the penalty for revocation. When scheduling succeeds, the scheduler needs to pay for agreements and penalties and $\delta - Prc(\mathcal{AG}) - \rho(\mathcal{AG})$ is a bonus for cost savings. $\beta\delta$ is another bonus for successful scheduling: it ensures that the scheduler gets some utility for succeeding, even it has spent all the budget $\delta$ for negotiation. Without this extra bonus, a scheduler that succeeds but spends all the budget gets zero utility, as if it had done nothing at all.

If scheduling fails, the scheduler receives no bonus but needs to pay for penalties. (Note that the maximum revocation penalty for any agreement is the full price.)

*PG* receives two types of tasks from the scheduler: negotiation tasks and revocation tasks. The scheduler has incomplete information about how *PG* will behave given each negotiation task. Given a resource $r$, $t(r)$ is the expected negotiation time for getting re-

source $r$, $c(r)$ is the price for getting resource $r$, and $p(r)$ is the probability that *PG* can get resource $r$ successfully. Without loss of generality, a resource in different plans has the same $t(r)$, $c(r)$, and $p(r)$. After receiving a revocation task, *PG* will 1) notify the other party involved in the agreement, 2) pay a penalty to the other party, 3) report to the scheduler. For ease of analysis, here we assume that 1) revocation tasks cost *PG* no time, 2) each negotiation task involves only one resource, and 3) *PG* can only negotiate for one resource at each time. Therefore, the scheduler can submit another task if and only if *PG* finishes the previous task.

After receiving plans from the planner, the scheduler has the following choices: 1) submit a negotiation task to *PG* if it can presumably earn positive utility by scheduling; 2) report "failure" to the planner if the scheduler finds that it could not schedule any plan within the reserve price $\delta$ and scheduling deadline $\tau$. The following observation shows the conditions under which the scheduler will return "failure" directly at the beginning of scheduling.

PROPOSITION 1. *If* $\min_{PL\in\mathcal{PS}}\sum_{r\in\mathcal{PL}} t(r) > \tau$ *or* $\min_{PL\in\mathcal{PS}}\sum_{r\in\mathcal{PL}} c(r) > \delta$, *the scheduler will directly return return failure information to the planner.*

PROOF. $\min_{PL\in\mathcal{PS}}\sum_{r\in\mathcal{PL}} t(r)$ is the minimum time needed to get all the resources in any plan. If $\min_{PL_i\in\mathcal{PS}}\sum_{r\in\mathcal{PL}} t(r) > \tau$, it is impossible for the scheduler to find a plan with agreements. Similarly, if $\min_{PL\in\mathcal{PS}}\sum_{r\in\mathcal{PL}} c(r) > \delta$, the scheduling will fail as $\min_{PL\in\mathcal{PS}}\sum_{r\in\mathcal{PL}} c(r)$ is the minimum money needed to return a plan with agreements. □

*Feasible* plans refer to the set of plans whose resource requirements can still be possibly satisfied by future negotiation. Assume *PG* fails to reach an agreement for resource $r$, all plans which include the resource $r$ will become infeasible. Therefore, the feasible resource set $\mathcal{PS}^t$ at scheduling round $t$ has the following property: $\mathcal{PS}^t \subseteq \mathcal{PS}^{t-1} \subseteq \mathcal{PS}$. After getting the feedback from the *PG* at scheduling round $t$, the scheduler has the following four choices:

- Submit another negotiation task to *PG*: If no plan's resource requirements has been satisfied and there is an opportunity to get all agreements for the resource requirements of one plan, the scheduler may submit a new negotiation task to *PG*. If one plan's resource requirements have been satisfied, the scheduler can still submit a negotiation task in order to satisfy the resource requirements of a better plan with lower cost.
- Revoke one or more agreements: As scheduling continues, some agreements may become unnecessary. For example, resource $r$ and $r'$ appears and only appears in plan $PL$. If *PG* fails to get resource $r$, plan $PL$ is unfeasible. Then the agreement for resource $r'$ becomes unnecessary. Unnecessary agreements should be revoked as early as possible to minimize penalty.
- Return a plan with agreements to the planner: Once all the resource requirements of one plan have been satisfied and there is no opportunity to satisfy the resource requirements of a better plan, the scheduler will return the plan with agreements to the planner and stop scheduling.
- Return "failure" to the planner: If the budget constraint or the deadline constraint makes the scheduler unable to satisfy the resource requirements of any plan, the scheduler will return "failure" to the planner and stop scheduling.

A naive way to calculate the optimal schedule is by exhaustive search. If there are $n$ resources in all the plans, the complexity of exhaustive search is $\mathcal{O}(n!)$. An alternative is to randomly choose a resource to negotiate during each scheduling round. Although the randomized approach has low complexity $\mathcal{O}(1)$, its scheduling results may be far from optimal. This paper presents a voting-based approach and a weight-based approach to allow tradeoffs between complexity and utility.

**Algorithm 1** Voting algorithm for the single-plan case

**Input:** Agreement set $\mathcal{AG}$, plan $PL$, deadline $\tau'$.
**Output:** A schedule $\Upsilon$.
**Data Structure:** $V(r)$ for each resource $r \in RT^t(\mathcal{PL})$.

```
1: Let Size = 1, V(r) = 0 for each resource r ∈ RT^t(PL).
2: repeat
3:   Size++;
4:   for each combination of resources R ⊆ RT^t(PL) where |R| = Size do
5:     if deadline has not approached then
6:       Find the best schedule Γ for the set of resource R;
7:       for each resource r ∈ R do
8:         if r is the i^th resource in Γ then
9:           V(r) = V(r) + |R| + 1 − i;
10:        end if
11:      end for
12:    end if
13:  end for
14:  Generate plan Υ in which resource r is behind resource r' if V(r) < V(r')
15: until scheduling deadline is approached or Size == |RT^t(PL)|
16: return Υ
```

# 4. VOTING-BASED ALGORITHM

As the complexity of exhaustive search factorially increases with the number of resources in the plans, it is infeasible to run the exhaustive search to find out the optimal schedule. To permit tradeoffs between the high complexity and utility, this section presents a voting-based approach which performs exhaustive searches on small resource sets and derives the negotiation schedule for large resource sets based on the search results.

## 4.1 Algorithm for Single-plan Case

We begin with a simpler version of the problem in which there is only one plan $PL$, i.e., $\mathcal{PS} = \{PL\}$. At each step of scheduling, the scheduler needs to find out the next resource to be submitted to the *PG* for negotiation. The set of resources having not been negotiated at scheduling round $t$ is $RT^t(PL) = \mathcal{PL} - \cup_{t'=0}^{t-1} T^{t'}$, where $T^{t'}$ is resource submitted at round $t'$. $RT^0(PL) = \mathcal{PL}$. $|RT^t(PL)| = |\mathcal{PL}| - t$ is the number of resources in $RT^t(PL)$. At round $t$, the scheduler needs to find the optimal schedule $\Lambda_o^t \in \Lambda^t$ in which $\Lambda^t$ represents all the possible schedules at round $t$ and $T^{t'}(\Upsilon)$ is the negotiation task at round $t'$ according to the schedule $\Upsilon \in \Lambda^t$. If negotiation for resource $T^{t-1}$ fails, $PL$'s resource requirements cannot be satisfied and the scheduler will stop scheduling and revoke all agreements made before.

Given a schedule $\Upsilon$ at round $t$, the expected utility while conducting the schedule is $U(\Upsilon) = U_{suc}(\Upsilon) - P_{fail}(\Upsilon)$ where $U_{suc}(\Upsilon)$ is the utility of the scheduler when it gets all resources and $P_{fail}$ is the penalty the scheduler needs to pay if scheduling fails. We have

$$U_{suc}(\Upsilon) = \prod_{r\in RT^t(PL)} p(r)\Big((1+\beta)\delta - \sum_{r\in\mathcal{PL}} c(r)\Big)$$

$$P_{fail}(\Upsilon) = \sum_{t'=t}^{|\mathcal{PL}|-1} p_{fail}^{t'}(\Upsilon) P_{fail}^{t'}(\Upsilon)$$

where $p_{fail}^{t'}(\Upsilon)$ is the probability that the scheduler fails to get $T^{t'}(\Upsilon)$ and $P_{fail}^{t'}(\Upsilon)$ is the penalty that the scheduler needs to pay if it fails to get resource $T^{t'}(\Upsilon)$:

$$p_{fail}^{t'}(\Upsilon) = \prod_{t''=t}^{t'-1} p(T^{t''}(\Upsilon))\Big(1 - p(T^{t'}(\Upsilon))\Big)$$

$$P_{fail}^{t'}(\Upsilon) = \sum_{t''=0}^{t-1} \rho\Big(c(T^{t''}), \sum_{t'''=t''+1}^{t-1} t(T^{t'''}) + \sum_{t'''=t}^{t'} t(T^{t'''}(\Upsilon))\Big) + \sum_{t''=t}^{t'-1} \rho\Big(c(T^{t''}), \sum_{t'''=t}^{t'} t(T^{t'''}(\Upsilon))\Big)$$

The optimization problem for the scheduler is to find the optimal negotiation schedule $\Lambda_o^t$ for the remaining resources $RT^t(PL)$ at round $t$, which can be formalized as follows:

$$\Lambda_o^t = \max_{\Upsilon \in \Lambda^t} \Big(U_{suc}(\Upsilon) - P_{fail}(\Upsilon)\Big)$$

It can be found that any schedule $\Upsilon \in \Lambda^t$ has the same $U_{suc}(\Upsilon)$ because changing the order of resources in any schedule $\Upsilon$ has no effect on $U_{suc}(\Upsilon)$. Thus, the schedule minimizing $P_{fail}(\Upsilon)$ would be the optimal schedule.

Let $|RT^t(\mathcal{PL})| = n$, then the complexity of finding the optimal schedule by exhaustive search is $\mathcal{O}(n!)$, which is computationally intractable. The intuition behind the voting-based approach is based on *order invariability*, which indicates the following: if resource $r$ appears before resource $r'$ in the optimal schedule for resource set $R'$, then resource $r$ appears before resource $r'$ in the optimal schedule for resource set $R''$ where $R' \subset R''$. Empirical evidence suggests that, in most cases (more than 90% in all our experiments), order invariability is valid. Given the deadline constraints and the high complexity of the problem, it is worthwhile to exploit the order invariability property when it almost applies.

Rather than exhaustively searching in the optimal schedule for remaining resources $RT^t(\mathcal{PL})$, in the voting-based approach, we only search the optimal schedule for a small set of resources $\mathcal{R} \subseteq RT^t(\mathcal{PL})$. Then we compute the schedule for $RT^t(\mathcal{PL})$ using the optimal schedules for all combinations of a smaller number of resources. Let $|\mathcal{R}| = q$, there are $\binom{n}{q}$ different combinations of resources of size $q$. The complexity of computing the optimal schedules for all $\binom{n}{q}$ combinations is $\mathcal{O}(\binom{n}{q} q!)$ ($\mathcal{O}(n^q)$ when q is constant). The complexity is reduced by

$$\frac{n!}{\binom{n}{q} q!} = (n-q)!$$

Algorithm 1 shows how to find the schedule for $RT^t(\mathcal{PL})$ using voting. The input is the current agreement set $\mathcal{AG}$, the plan $PL$, and the deadline $\tau'$, which is the maximum time the algorithm can run. The output of the algorithm is a schedule $\Upsilon$ for $RT^t(\mathcal{PL})$.

$V(r)$ denotes how many "votes" $r$ receives; it is used to represent how desirable it is to negotiate resource $r$ earlier. The resource with the highest $V(r)$ will be negotiated first. $Size$ represents the number of resources which will be used to run exhaustive search. At the beginning, $Size$ is set to 2 and the scheduler will first find the optimal schedule of every pair of resources. Then $Size$ will be set to 3 and all combinations of three resources are searched, then $Size$ is set to 4, 5, and so on.

For each combination of resources $\mathcal{R}$, the scheduler will search all possible schedules and choose the one with the highest utility as the optimal schedule $\Gamma$. The $i$th resource in an optimal schedule of $q$ resources receives a "vote" of $q+1-i$ and $V(r)$ is increased by the "votes" from all combinations where $r$ appears.

After scheduling all the combinations of $Size$ resources, the output schedule $\Upsilon$ will be updated. if $V(r) < V(r')$, resource $r$ will be negotiated later than resource $r'$ in the plan $\Upsilon$. The algorithm will stop when the scheduling deadline is reached or all possible schedules have been searched, i.e., $Size = |RT^t(\mathcal{PL})|$. When the algorithm stops, the latest schedule $\Upsilon$ will be returned.

The algorithm is an anytime algorithm in the sense that with more time, the algorithm will search the optimal schedule of a larger number of negotiation tasks with the increase of $Size$. With more time, the schedule returned by the algorithm is better.

**Algorithm 2** Voting-based algorithm

**Input:** Plan set $\mathcal{PS}$, deadline $\tau$, reserve price $\delta$.
**Output:** A plan $PL^o$ with agreement or "failure".
**Data Structure:** Task set $\mathcal{T}$, agreement set $\mathcal{AG}$, feasible plan set $\mathcal{PS}^t$.

```
Let t = 0, T = ∅, AG = ∅, PS^0 = PS, fail=false, PL^o = ∅.
repeat
    Step 1: initialization (Algorithm 3)
    Step 2: revoke agreements and stop negotiation:
    if PL^o! =null then
        revoke agreements in AG which is not needed for PL^o
        return PL^o with agreements
    else
        if |PS^t| = 0 then
            revoke all agreements in AG
            return "failure"
        else
            revoke agreements which are not needed in any plan in PS^t
        end if
    end if
    Step 3: submit a proposal
    for each resource r ∈ RT^t(PS^t) which has not been negotiated do
        compute its weight w(r)
    end for
    Let the task at t be max_{r∈RT^t(PS^t)} w(r) and submit it to PG;
    t++;
    PS^t = PS^{t-1};
until false
```

## 4.2 Algorithm for the Multi-plan Case

If there are more than one feasible plan, we can run the voting algorithm for each plan, and then compute the overall weight $w(r)$ for each resource $r \in RT^t(\mathcal{PS}^t)$ at round $t$ where $RT^t(\mathcal{PS}^t) = \cup_{PL \in \mathcal{PS}^t} RT^t(\mathcal{PL})$ is the set of all possible negotiation tasks. Algorithm 2 describes the procedure to determine the negotiation task at each round $t$. The input of the algorithm is the set $\mathcal{PS}$ of plans submitted by the planner, deadline $\tau$, and reserve price $\delta$. The output is the plan $PL^o$ with agreement if scheduling succeeds or "fail-

**Algorithm 3** Initialization

1: **if** $t == 0$ **then**
2: **for** each $PL \in \mathcal{PS}^t$ **do**
3: remove $PL$ from $\mathcal{PS}^t$ if $\sum_{r\in\mathcal{PL}} t(r) > \tau$ or $\sum_{r\in\mathcal{PL}} c(r) \geq (1+\beta)\delta$
4: **end for**
5: **else**
6: **if** *PG* fails to get resource $T^{t-1}$ **then**
7: remove plans which need resource $T^{t-1}$ from $\mathcal{PS}^t$
8: **for** each $PL \in \mathcal{PS}^t$ **do**
9: remove $PL$ from $\mathcal{PS}^t$ if $\tau - \sum_{t'=0}^{t-1} t(T^{t'})$ is not enough to get the resources having not been negotiated in plan $PL$
10: **end for**
11: **else**
12: **if** some plans' resource requirements have been satisfied **then**
13: Let $PL^o$ be the plan which minimize the sum of the cost for the agreements needed by the plan and the penalty for the remaining agreements
14: **end if**
15: **end if**
16: **end if**

ure" otherwise. During scheduling, the scheduler needs to maintain the following data structure: task set $\mathcal{T}$, agreement set $\mathcal{AG}$, feasible plan set $\mathcal{PS}^t$. Algorithm 2 includes the following three main steps:

At the beginning of each round, the scheduler updates its data structure given the last round's negotiation result. If $t = 0$, these plans which cannot satisfy the budget and deadline constraints will be removed from $\mathcal{PS}^t$. When $t > 0$, if *PG* fails to get resource $T^{t-1}$, it will first remove plans which need resource $T^{t-1}$ from $\mathcal{PS}^t$. Then it will check whether the remaining scheduling time $\tau - \sum_{t'=0}^{t-1} t(T^{t'})$ is enough to get the resources having not been negotiated in plan $PL$. It not, it will remove $PL$ from $\mathcal{PS}^t$. If *PG*'s negotiation at round $t-1$ is successful, some plans' resource requirements may have been satisfied. Assume the satisfied plans are $\mathcal{PS}_s \subseteq \mathcal{PS}^t$ and a plan $PL \in \mathcal{PS}_s$ needs a set of agreements $\mathcal{AG}(PL)$. The following plan will be chosen as $PL^o$:

$$PL^o = \min_{PL\in\mathcal{PS}_s} \left( \sum_{r\in PL} c(r) + \sum_{Ag\in\mathcal{AG}-\mathcal{AG}(PL)} \rho(Prc(Ag), \sum_{t'=0}^{t-1} t(T^{t'}) - T_{mk}(Ag)) \right)$$

In other words, $PL^o$ is the plan that minimizes the sum of the cost for the agreements needed by the plan and the penalty for revoking the remaining agreements.

The second step is revoking unnecessary agreements and checking whether to stop scheduling. There are three situations: 1) $PL^o! = null$, i.e., one plan's resource requirements have been satisfied. The scheduler will revoke all unnecessary agreements $\mathcal{AG}-\mathcal{AG}(PL^o)$ and return plan $PL^o$ with agreements $\mathcal{AG}(PL^o)$ to the planner. 2) $|\mathcal{PS}^t| = 0$, which means that scheduling fails because there is no satisfiable plan left. The scheduler will revoke all agreements $\mathcal{AG}$ and return "failure" to the planner. 3)$PL^o = null$ and $|\mathcal{PS}^t| > 0$, which indicates scheduling will continue and the scheduler will revoke agreements which are not needed in any plan in $\mathcal{PS}^t$.

The most important part is deciding which resource to negotiate at round $t$. For each resource $r \in RT^t(\mathcal{PS}^t)$, its wight $w(r)$ represents how desirable it is to negotiate it first. That is, the negotiation task during round $t$ is the resource with the highest weight, i.e.,

$$T^t = \max_{r\in RT^t(\mathcal{PS}^t)} w(r)$$

in which $w(r)$ is given by:

$$w(r) = \sum_{PL\in\mathcal{PS}^t} w(r, PL)$$

where $w(r, PL)$ is the weight of resource $r$ in plan $PL$.

To compute $w(r, PL)$, the scheduler will first run the voting algorithm to generate the schedule for $RT^t(\mathcal{PL})$. To get the schedule for the remaining resource $RT^t(\mathcal{PL}) \subseteq \mathcal{PL}$, the scheduler needs to decide the deadline $\tau'$ to run Algorithm 1. Let $\tau''$ be the remaining scheduling time and it follows that $\tau'' \geq \max_{PL\in\mathcal{PS}^t} \sum_{r\in RT^t(\mathcal{PL})} t(r)$ as otherwise, the plan $PL$ will not be feasible. Here we use a heuristic to decide the value of $\tau'$:

$$\tau' = \frac{\tau'' - \frac{\sum_{PL\in\mathcal{PS}^t}\sum_{r\in RT^t(\mathcal{PL})} t(r)}{|\mathcal{PS}^t|}}{|\cup_{PL\in\mathcal{PS}^t} RT^t(\mathcal{PL})|}$$

where $\frac{\sum_{PL\in\mathcal{PS}^t}\sum_{r\in RT^t(\mathcal{PL})} t(r)}{|\mathcal{PS}^t|}$ is the average negotiation time for each remaining plan and $|\cup_{PL\in\mathcal{PS}^t} RT^t(\mathcal{PL})|$ is the number of resources in all the remaining plans.

After getting the schedule $\Upsilon$ for $RT^t(\mathcal{PL})$ of the plan $PL$ using Algorithm 1, the weight $w(r, PL)$ of resource $r \in RT^t(\mathcal{PL})$ is

$$w(r, PL) = \begin{cases} \frac{|RT^t(\mathcal{PL})|+1-i}{\sum_{r\in PL} c(r)} & \text{if } r \text{ is the } i^{th} \text{ resource in } \Upsilon \\ 0 & \text{otherwise} \end{cases}$$

It can be found that the higher the order of resource $r$ in the scheduler $\Upsilon$ for $PL$, the higher the value of $w(r, PL)$. If $r$ appears first in all the schedulers for plans $\mathcal{PS}^t$, $r$ will be chosen as $T^t$.

It can be found in Algorithm 2 that the scheduler will stop scheduling once the resource requirements of one plan are satisfied. This is partly because in our approach, a plan with lower cost has a higher chance to be satisfied before a plan with higher cost. In addition, given a plan with agreements, the scheduler needs to revoke agreements made before if it finds another plan with agreements. Moreover, uncertainty about negotiation results makes it possible that the scheduler fails to satisfy the resource requirements of another plan but pays more penalties.

Let $n = |\cup_{PL\in\mathcal{PS}} \mathcal{PL}|$ be the number of resources in all plans $\mathcal{PS}$ and $m = |\mathcal{PS}|$ be the number of plans. If $Size = q$, the complexity of using the voting-based approach to determine the negotiation task at round $t$ is $\mathcal{O}(m\binom{n}{q}q!) = \mathcal{O}(mn!/(n-q)!) = \mathcal{O}(mn^q)$. We can find that even we choose very small voting depth $q$, the complexity of the voting-based approach is still high. In the next section, we present a weight-based approach, which has a lower complexity of $\mathcal{O}(mn)$.

# 5. WEIGHT-BASED APPROACH

This section presents a weight-based approach. Rather than doing an exhaustive search on a small scale, the weight of each resource is computed directly by considering the characteristics of the multiple plans and resources in these plans.

## 5.1 Analysis for the Single-plan Case

Different resources have different prices, probabilities of successful negotiation, and negotiation times. It would help the scheduler to find the optimal schedule if we can find how those resource properties will affect the optimal schedule.

PROPOSITION 2. *Resources with higher prices should be negotiated later.*

PROOF. We prove it by comparing the utility while changing the order of two resources $T^{t'}(\Upsilon) = r$ and $T^{t'+1}(\Upsilon) = r'$ in the schedule $\Upsilon$. Assume the new schedule after changing the order of the two resources is $\Gamma$. For ease of analysis, assume resources $r$ and $r'$ have the same probability of successful negotiation and negotiation time, but with different prices. Then we have:

$$\begin{aligned}
U(\Upsilon) - U(\Gamma) &= U_{suc}(\Upsilon) - P_{fail}(\Upsilon) - U_{suc}(\Gamma) + P_{fail}(\Gamma) \\
&= P_{fail}(\Gamma) - P_{fail}(\Upsilon) \\
&= \sum_{t''=t'}^{|\mathcal{PL}|-1} \Big( p_{fail}^{t''}(\Gamma) P_{fail}^{t''}(\Gamma) - p_{fail}^{t''}(\Upsilon) P_{fail}^{t''}(\Upsilon) \Big) \\
&= \alpha \prod_{t''=t}^{t'-1} p\Big(t(T^{t''}(\Upsilon))\Big) p(r) t(r) \Big(c(r') - c(r)\Big) \\
&\Big(1 - p(r')\Big) + \alpha \sum_{t''=t'+2}^{|\mathcal{PL}|-1} p_{fail}^{t''}(\Upsilon) t(r) \Big(c(r') - c(r)\Big)
\end{aligned}$$

It can be found that if $c(r') > c(r)$, $U(\Upsilon) > U(\Gamma)$; otherwise $U(\Upsilon) < U(\Gamma)$. This indicates that cheaper resources should be negotiated before the expensive resources. □

This observation corresponds to the intuition that if we negotiate a resource with a higher price first, the revocation penalty later will also be higher when the agreement becomes unnecessary as the penalty increases with the price in the agreement.

PROPOSITION 3. *Resources with higher success probabilities should be negotiated later.*

PROOF. Similarly, assume there are two schedules $\Upsilon$ and $\Gamma$ which are the same except that in $\Upsilon$, $T^{t'}(\Upsilon) = r$, $T^{t'+1}(\Upsilon) = r'$. In the schedule $\Gamma$, $T^{t'}(\Gamma) = r'$, $T^{t'+1}(\Gamma) = r$. Assume $r$ and $r'$ are only different in the probabilities of successful negotiation. We have

$$\begin{aligned}
U(\Upsilon) - U(\Gamma) &= U_{suc}(\Upsilon) - P_{fail}(\Upsilon) - U_{suc}(\Gamma) + P_{fail}(\Gamma) \\
&= P_{fail}(\Gamma) - P_{fail}(\Upsilon) \\
&= \sum_{t''=t'}^{|\mathcal{PL}|-1} \Big( p_{fail}^{t''}(\Gamma) P_{fail}^{t''}(\Gamma) - p_{fail}^{t''}(\Upsilon) P_{fail}^{t''}(\Upsilon) \Big) \\
&= \alpha \prod_{t''=t}^{t'-1} p\Big(t(T^{t''}(\Upsilon))\Big) \Bigg( \sum_{t'''=0}^{t'-1} c(t(T^{t'''}(\Upsilon) t(r) \\
&\Big(p(r') - p(r)\Big) + c(r') t(r) \Big(p(r') - p(r)\Big) \Bigg)
\end{aligned}$$

It can be found that if $p(r') > p(r)$, $U(\Upsilon)$ is better than $U(\Gamma)$, which indicates that the resources with higher success rates should be negotiated first. □

This observation corresponds to the intuition that if we negotiate resources with lower success probabilities later, the scheduler will have higher probability of failing to obtain those resources and will have to revoke agreements made before, thus paying more penalties. Therefore, we would like to first negotiate resources which are difficult to get, in order to reduce such revocation penalties.

## 5.2 Analysis for the Multi-plan Case

The two observations above are helpful while developing scheduling algorithms. However, different resources have different success probabilities and prices. Thus, the scheduler needs to make tradeoffs to solve the conflicts between different properties. In addition, the two observations are valid for the single-plan case. In the multiple-plan case, in addition to success probability and price, the following factors should also be considered:

- Appearance frequency. Assume there are two resources $r$ and $r'$. $r$ appears in most plans and resource $r'$ appears much less than $r$. Assume there is no big difference in $r$ and $r'$'s other properties like price. It is intuitive to negotiate resource $r$ first because if *PG* cannot get resource $r$, *PG* does not need to consider those plans which need resource $r$ anymore. An extreme situation is that resource $r$ appears in all the plans. Obviously, we need to get resource $r$ first. If *PG* fails to get resource $r$, scheduling stops. If the scheduler lets *PG* negotiate for $r'$ first and *PG* successfully gets resource $r'$, then the scheduler later needs to pay for revoking the agreement about $r'$ when *PG* fails to get resource $r$.

- Plan cost: The cost of a plan $PL$ is $C(PL) = \sum_{r \in \mathcal{PL}} c(r)$. The costs of different plans may vary. Regardless of other factors like negotiation time, the scheduler would like to satisfy the resource requirements of the plan with lower cost first as the scheduler will try to maximize its utility, which will decrease with the increase of the plan cost.

Based on our observations and intuitions, we propose a heuristic to compute the weight $w(r, PL)$ of resource $r$ as follows

$$w(r, PL) = f(\sum_{r \in PL} c(r), p(r), |RT^t(\mathcal{PL})|)$$

where $\sum_{r \in PL} c(r)$ is the cost of the plan $PL$, $|RT^t(\mathcal{PL})|$ is the number of resources to be negotiated in plan $PL$. $w(r, PL)$ will satisfy the following properties:

- $w(r, PL)$ decreases with $\sum_{r \in PL} c(r)$. That is, if the cost of the plan is high, any resource of this plan has a lower probability to be negotiated first.

- $w(r, PL)$ decreases with $p(r)$, which corresponds to our previous proposition that the resource with lower success probability should be negotiated first.

- $w(r, PL)$ decreases with $|RT^t(\mathcal{PL})|$. With fewer remaining resources to be negotiated, the scheduler can know whether it can satisfy the resource requirements of the plan in shorter rounds. Thus, it is reasonable to negotiate plans with less resources first because those plans seem "hopeful".

$w(r)$ is the sum of the weight of resource $r$ in all the plans. If the resource $r$ appears more in those plans, it will have a higher weight. Therefore, it has a higher probability of being negotiated first. In other words, frequently appearing resources will be negotiated first. An example function to compute $w(r, PL)$ is:

$$w(r, PL) = \begin{cases} \frac{1}{\sum_{r \in PL} c(r) p(r) |RT^t(\mathcal{PL})|} & \text{if } r \in RT^t(\mathcal{PL}) \\ 0 & \text{otherwise} \end{cases}$$

Given $w(r, PL)$ of each resource in plan $PL$, we can compute the weight $w(r)$ of each resource $r \in RT^t(\mathcal{PS}^t)$. Then we can use Algorithm 2 to determine the negotiation task at each round.

**Table 2: Variables**

| *Variables* | Values |
|---|---|
| *Resource prices* | $[1, 200]$ |
| *Negotiation time* | $[1, 50]$ |
| *Success probability* | $[0, 1]$ |
| *Number of plans* | $[2, 30]$ |
| *Number of resources in a plan* | $[2, 10]$ or $[2, 20]$ |
| *Voting depth* | 3 or $[2, 10]$ |

The complexity of the weight-based approach at each scheduling round is $\mathcal{O}(nm)$ where $n = |\cup_{PL \in \mathcal{PS}} \mathcal{PL}|$ is the number of resources in all the $m = |\mathcal{PS}|$ plans. Obviously, the weight-based approach has much lower complexity than the voting-based approach, even when the voting depth is small. When the voting depth is q, the complexity of the voting based approach is $\mathcal{O}(n^q m)$, which is higher than $\mathcal{O}(nm)$.

# 6. EXPERIMENTAL RESULTS

We implemented a simulation testbed consisting of a manager, planner, scheduler and proposal generator to evaluate the performance of our proposed algorithms. The manager generates resources and randomly determines their parameters (e.g., negotiation time, price, probability of success). The planner generates plans and determines their parameters (e.g., reserve price and scheduling deadline). *PG* sequentially negotiates for resources and the negotiation results follow the distributions defined by the manager. The scheduler uses different scheduling algorithms to submit negotiation tasks to *PG* and return scheduling results to the planner. In order to demonstrate the performance of the proposed two approaches by comparison, a randomized approach was also evaluated in which the scheduler randomly chose a resource to negotiate. Additionally, exhaustive search is used to calculate the optimal schedule for scenarios with small numbers of resources where exhaustive search is tractable.

We performed a series of experiments to compare the performance of the weight-based and voting-based approaches. The parameters are given in Table 2. Resources were subjected to different resource prices, negotiation times, and success probabilities. In the experiments, the resource price is randomly selected from $[1, 200]$, the negotiation time of each resource from $[1, 50]$, and the success probability of each resource from $[0, 1]$. The number of alternative plans submitted to the scheduler is between $[2, 30]$. Each plan is subject to different numbers of resources, reserve prices, and deadlines. The number of resources in each plan is randomly chosen from $[2, 20]$. The reserve price $\delta$ is defined as $a \min_{PL \in \mathcal{PS}} \sum_{r \in \mathcal{PL}} c(r)$, where $a$ randomly selected from $[1, 3]$. Similarly, the deadline $\tau$ is defined as $b \min_{PL \in \mathcal{PS}} \sum_{r \in \mathcal{PL}} t(r)$, where $b$ is randomly selected from $[1, 3]$. For the voting algorithm, different voting depths may result in different scheduling performance. Due to the high complexity of the exhaustive search, the voting depth was set to 3 in all experiments except the one explicitly varying that parameter. $\beta$ was set to 0.1 in the experiments, but we found that varying this parameter within a small range $(0.1 - 0.5)$ did not significantly affect the results.

After each experiment, we measure the utility value of each scheduling result. As we evaluated algorithms in different environments, we normalize the utilities $u \in [u_{\min}, \delta]$ of different experiments into the same range $[0, 1]$ where $u_{\min} < 0$ is the lowest utility the scheduler may receive when the scheduling fails and $\delta$ is the budget. The normalized utility $u'$ of an experiment is given as:

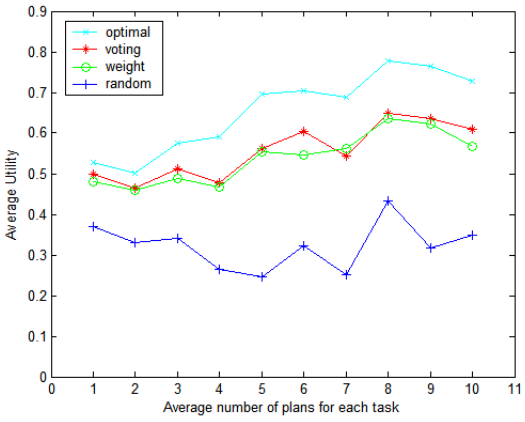


**Figure 1: Average utility as a function of the number of plans**

$$u' = \frac{u - u_{\min}}{\delta - u_{\min}}$$

Figs. 1-3 show some representative experimental results gathered from a series of experiments in different environment settings. Each experiment consists of the average of 200 runs.

*Observation 1*: High complexity precludes computing the optimal schedule when there are a large number of resources. Fig. 1 shows the performance of different scheduling algorithms when the number of resources is no larger than 10. We can find that the voting-based approach and the weight-based approach achieve much higher utility (50% on average), sometimes more than 100% higher, than the randomized approach. The average utility obtained by the optimal schedule is, on average, around 10% higher than that of the two proposed algorithms.

We also ran experiments when the number of resource of each plan is between 2 and 20. We observed similar improvements of both heuristics over the randomized approach. Regardless of the number of resources, the voting-based one also achieves slightly higher (6%) utility than the weight-based one on average.

*Observation 2*: Experimental results in Fig. 2 suggest that the voting-based approach and the weight-based approach still perform better than the randomized approach as the average number of resources in each plan increases. Moreover, with more resources in each plan, the advantage of our heuristics increases, which corresponds to the intuition that the totally randomized approach cannot do well in much more complex environments.

*Observation 3*: Fig. 3 depicts the change of the CPU time (in milliseconds) and average utility for the voting-based algorithm. In this experiment, there are 12 resources and 3 plans. The computation overhead is shown on a log scale in Fig. 3. We can find that the computation overhead of the voting-based approach almost exponentially increases with the increase of voting depth. Moreover, with the increase of voting depth, the increase of average utility is much slower than the increase of computation overhead. The results imply that in resource bounded environments, the weight-based approach is a better selection.

# 7. CONCLUSION

This paper studies the scheduling problem of finding the optimal negotiation schedules given a set of candidate plans. The problem studied in this paper is complex due to multiple choices and uncertainty. We utilize the characteristics of all the candidate plans to

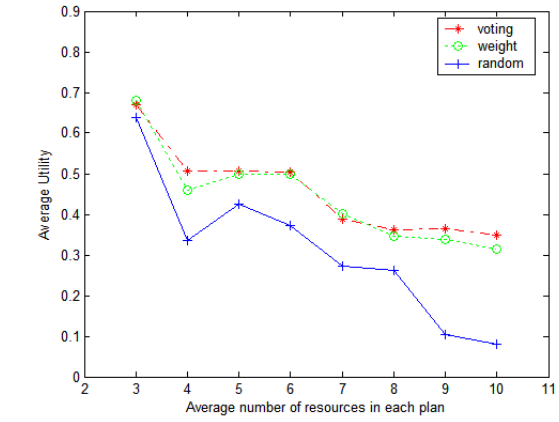


**Figure 2: Average utility as a function of the number of resources in plans**

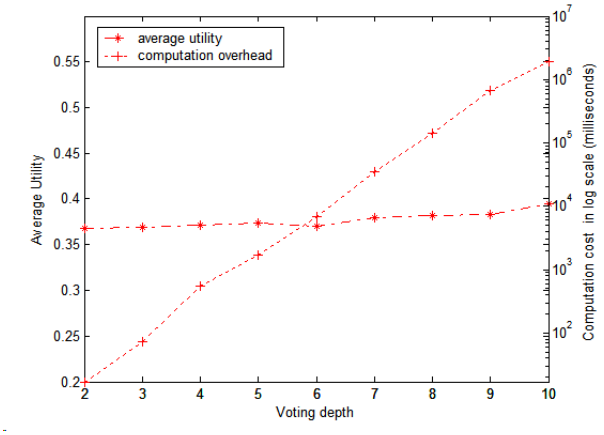


**Figure 3: Average utility and computing overhead as functions of voting depth. Overhead is shown on a log scale**

decide the order of resources to negotiate. We provide both voting-based and weight-based algorithms to determine the negotiation schedule. The voting-based approach has higher computation overhead than the weight-based approach. Experimental results suggest that the proposed approaches achieved 50% higher utility than the randomized approach and are within almost 90% of that of the optimal results with reasonable plan sizes.

Experimental results suggest that the voting-based approach is slightly better (6%) than the weight-based approach in average utility. But the voting-based approach has a higher complexity, especially when the voting depth is large. The weight-based approach is more suitable for "heavy" scheduling problems in which there are many plans and each plan consists of a large number of resources. If there are a small number of resources, the voting-based approach is more suitable. For resource-bounded environments, the weight-based approach is more suitable as it is more efficient in saving time than the voting-based approach. We can choose the voting-based approach when there is ample scheduling time as the average utility of the voting-based approach is higher than that of the weight-based approach and the performance of the voting-based approach improves with larger voting depth.

In general, the proposed heuristics can be applied in scheduling for resource acquisition in many other application domains, such as multi-agent manufacturing systems, autonomic and service oriented computing, dynamic web/grid service composition, virtual chain management, workflow, and enterprise integration.

Finally, a future agenda of this work includes:

- This work assumes static environments where all the plans are sent to the scheduler before the scheduling starts. This assumptions can be loosened by considering dynamic scenarios in which the scheduler may receive more plans during scheduling.
- We can consider other types of penalty functions than linear functions. For example, we can consider a grace period during which there is no revocation penalty.
- Plans may have different reserve prices to indicate their priority, which makes the scheduling problem more complex.
- Our experiments thus far focused on scenarios ranging from low to moderate complexity, but we wish to investigate much larger problems. Initial experiments with more demanding plans (fixing all plans to have 30 alternatives with 30 resources apiece) found that both the voting-based and weight-based heuristics have significantly lower utilities, due to the number of times planning fails; voting with a depth of two improves the average utility from 16% to 22% but takes 10 times as long (123ms).

## Acknowledgements

We thank Mike Branson, Brad Fawcett, Anton Riabov, Zhen Liu and Cathy Xia for helpful comments in the development of this work. Special thanks are due Brad Fawcett for implementing the Proposal Generator in our prototype system.

## 8. REFERENCES

[1] A. Andrieux et al. Web Services Agreement Specification (WS-Agreement), Version 2006/07. GWD-R (Proposed Recommendation), Grid Resource Allocation Agreement Protocol (GRAAP) WGGRAAP-WG, July 2006.

[2] M. Branson, F. Douglis, B. Fawcett, Z. Liu, A. Riabov, and F. Ye. CLASP: collaborating, autonomous data stream processing systems. In *Proc. of the ACM/IFIP/USENIX 8th International Middleware Conference*, pages 348–367, 2007.

[3] E. Deelman et al. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[4] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[5] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: prospects, methods and challenges. *Int. Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

[6] S. Kraus. *Strategic Negotiation in Multiagent Environments*. MIT Press, Cambridge, MA, 2001.

[7] R. H. Mohring, A. S. Schulz, and M. Uetz. Approximation in stochastic scheduling: the power of lp-based priority policies. *Journal of the ACM*, 46(6):924–942, Nov. 1999.

[8] J. Ni-Mora. *Encyclopedia of Optimization*, volume V, chapter Stochastic scheduling, pages 367–372. Kluwer, 2001.

[9] A. Riabov and Z. Liu. Planning for stream processing systems. In *Proc. of the Twentieth National Conference on Artificial Intelligence*, pages 1205–1210, 2005.