Journal of Universal Computer Science, vol. 19, no. 3 (2013), 282-300
submitted: 16/3/12, accepted: 28/1/13, appeared: 1/2/13

Co-Allocation with Collective Requests in Grid Systems

Matija Cankar, Matej Artač, Marjan Šterk
(XLAB d.o.o., Ljubljana, Slovenia
{matija.cankar, matej.artac, marjan.sterk}@xlab.si)

Uroš Lotrič, Boštjan Slivnik
(University of Ljubljana, Slovenia
{uros.lotric, bostjan.slivnik}@fri.uni-lj.si)

Abstract: We present a new algorithm for resource allocation in large, heterogeneous grids. Its main advantage over existing co-allocation algorithms is that it supports collective requests with partial resource reservation, where the focus is on better grid utilisation. Alongside the requests that must be fulfilled by each resource, a collective request specifies the total amount of a required resource property without a strict assumption with regard to its distribution. As a consequence, the job becomes much more flexible in terms of its resource assignment and the co-allocation algorithm may therefore start the job earlier. This flexibility increases grid utilisation as it allows an optimisation of job placement that leads to a greater number of accepted jobs. The proposed algorithm is implemented as a module in the XtreemOS grid operating system. Its performance and complexity have been assessed through experiments on the Grid'5000 infrastructure. The results reveal that in most cases the algorithm returns optimal start times for jobs and acceptable, but sometimes suboptimal resource sets.

Key Words: Resource co-allocation, Grid computing, Parallel applications, Concurrency, Advance reservations

Category: C.2.4, C.2.m, C.3

1 Introduction

The ever-growing demand for more computer power has spurred the development of distributed computing, which has resulted in grid and cloud computing. With the rise of cloud computing [García-Peñalvo et al. 2012], it may appear that grids are no longer of interest. However, cloud computing introduces a virtualisation overhead by reducing the efficiency of computation and increasing latencies [Younge et al. 2011], so grids are still used when high-performance computation is needed.

Grid services ensure that jobs submitted into the grid are assigned proper resources and maintain high utilisation of the system at the same time. Services include modules for scheduling, resource provisioning and/or resource co-allocation [Netto and Buyya 2010]. In this paper we assume that the first two services, essential to any system, are already in place. Our focus is on the co-allocation service, which offers simultaneous access to resources hosted by autonomous providers [Czajkowski et al. 1999].

In homogeneous grids co-allocation modules operate with uniformly configured resources and usually handle requests such as "for a two-hour job, find ten resources, each with at least 4 GB of memory". This request might be too strict for heterogeneous grids with non-uniformly configured resources. It is very likely that a user would be satisfied with a request such as "for a two-hour job, find ten resources that together have at least 40 GB of memory". The latter request is more flexible and can lead to the job being started earlier and, at the same time, to better system utilisation.

We distinguish between two types of requests: *simple requests* consist only of requirements for each independent resource, while *collective requests* consist of at least one requirement for which the total quantity is specified regardless of its distribution.

Simple requests can easily be serviced by well-known co-allocators such as Gara [Foster et al. 1999], JSS [Elmroth and Tordsson 2007], HARC [MacLaren 2007], OAR [Nicolas et al. 2011], KOALA [Mohamed and Epema 2008] and GridARS [Takefusa et al. 2007]. The approach of describing requests with looser constraints used by [Liu and Foster 2003] is general enough to handle collective requests. However, their resource matching implemented as a constraint satisfaction problem does not focus on system utilisation. Another efficient online co-allocation algorithm which only handles simple requests is based on range searches to identify available resources simultaneously [Castillo et al. 2009]. If resources can be partially reserved, provisioning of resources is even more flexible. PlanetLab [Chun et al. 2003] enables sharing of resources but lacks the automatic co-allocation module common to grids. OAR [Nicolas et al. 2011] provides a kind of resource sharing between multiple users, but only within one reservation. More advanced partial reservation is supported by XtreemOS [Cortes et al. 2008, Nou et al. 2010].

The majority of the co-allocators cited here have deficiencies that we believe can be overcome if co-allocation supports collective requests with partial resource reservation and uses local optimisation of job placement. This approach ensures that fewer jobs are rejected and accepted jobs are started earlier. The basic idea behind the approach, with preliminary results, was presented in [Cankar et al. 2010].

Section 2 of the paper includes a formal definition of the co-allocation problem. Section 3 explains the proposed co-allocation algorithm and Section 4 describes the experimental set-up and the main results. The paper concludes with the main findings and some ideas for future work.

2 Co-allocation

The problem of co-allocation is the selection of a set of resources that fulfil a given request. A *resource* is the smallest unit that can function independently and is described by a list of *resource properties*. Quantitative resource properties are specified in standard units, while qualitative resource properties are described with predefined labels. In a typical grid system, a resource is a computer node that can be described by quantitative resource properties such as the amount of memory and the CPU frequency, and by qualitative resource properties such as the CPU architecture and the version of the operating system. Additionally, a list of resource properties can include the price of the computation, latencies or bandwidths between the resources and the data store. Each grid system has its own list of resources and a pre-defined list of resource properties. The set of all quantitative resource properties on the grid is denoted by $\mathcal{P}$ and the amount of resource property P_i available on resource R_j is denoted by p_{ij}. The amount of resource property P_i reserved at time t on resource R_j is given by *timetable* $T_{ij}(t)$.

A *job* consists of the application, the input data, and the request that describes a relevant resource set. In our case a *collective request* consists of:

- the job duration t_{res} for which the reservation is needed,
- the number of resources N to be allocated,
- the earliest and/or latest possible start time, t_e and t_l,
- a set of values e_i, where e_i specifies the minimum amount of quantitative resource property P_i available at each resource,
- a set of values d_i, where d_i specifies the minimal amount of quantitative resource property P_i over all allocated resources, and
- a set of qualitative requirements.

It is mandatory to set t_{res} and N. If t_e and t_l are not specified, t_e equals the time of submission and $t_l = t_e + t_C$, where t_C is a predefined time constant. If e_i is undefined, a predefined small positive value of resource property is used. If d_i is undefined, a request becomes a *simple request* where $d_i = Ne_i$.

An *admissible solution* to the co-allocation problem consists of the start time t_s and a corresponding set of exactly N resources that satisfy the request and are available in a time slot defined by the interval $[t_s, t_s + t_{\mathrm{res}}]$.

The utilisation of each quantitative resource property P_i at time t on a set of resources $\mathcal{R}$ is defined as a ratio between the reserved amount of resource property and the total amount of the resource property

$$u_i(\mathcal{R}, t) = \frac{\sum_{R_j \in \mathcal{R}} T_{ij}(t)}{\sum_{R_j \in \mathcal{R}} p_{ij}} \quad . \tag{1}$$

Ideal utilisation is achieved when $u_i(\mathcal{R}, t)$ equals 1 while values smaller than 1 indicate an excessive amount of the resource property P_i. Values larger than 1 characterise non-permissible situations, utilising more resources than available. To ensure that resources are utilised well, values smaller than but close to 1 are preferred. We have expressed overall utilisation of resources $\mathcal{R}$ in terms of the *utilisation factor*

$$U(\mathcal{R}, t) = \prod_{P_i \in \mathcal{P}} u_i(\mathcal{R}, t) \quad . \tag{2}$$

We chose the multiplication of utilisations because it favours admissible solutions with high $u_i(\mathcal{R}, t)$ more intensively than summation. The utilisation factor therefore encourages the placing of new jobs on resources with a higher utilisation rate. This can lead to a smaller number of running resources and consequently some energy savings.

The admissible solution is *optimal* if it has the earliest start time and no other admissible solution with the same start time has a higher utilisation factor.

Even though the utilisation factor is obtained by multiplying utilisations, optimal requirement fulfilment is still an NP-complete problem. The proof is given in the Appendix.

3 The Co-allocation Algorithm

Based on the request the algorithm outputs a list of resources selected for the job, the reservation time, and the updated timetables. If a solution is not found, the list of resources is empty and the reservation time is undefined.

The proposed algorithm consists of six phases: in Phase I the algorithm requests an initial set of resources from the resource discovery system; in Phase II it checks whether the initial set of resources provides a sufficient amount of resource properties; in Phase III it constructs a set of potential start times; in Phase IV it examines the set of start times and tries to find the earliest admissible solution; in Phase V it improves the admissible solution; finally, in Phase VI, it reserves the resources and forms the output.

3.1 Phase I — Resource Discovery

The algorithm selects a set $\mathcal{R}_{\text{job}}$ of resources that comply with all requirements regardless of the timetables d_i, and passes the set to Phase II. For better chances of success the algorithm requires $M \cdot N$ resources from a resource discovery

Phase II: Resource candidates examination

foreach $P_i \in \mathcal{P}$ **do**
  let $R_{\omega(1)}, R_{\omega(2)}, \ldots, R_{\omega(M \cdot N)}$ be resources of $\mathcal{R}_{\text{job}}$ sorted by P_i in descending order
  if $\sum_{l=1}^{N} p_{i\omega(l)} < d_i$ **then**
    $M = 2M$
    go to Phase I
  end
end

service, where $M \in [M_{\text{init}}, M_{\text{max}}]$ for $M_{\text{init}} \geq 1$. Initially M is set to M_{init} but can be increased on return from unsuccessful subsequent phases. Larger values of M increase the search space and the possibility of finding a solution, but at the same time they increase the algorithm response time.

If at least N resources are selected, the algorithm proceeds to Phase II. Otherwise, a solution cannot be constructed, the job request is rejected, and the algorithm terminates. The algorithm may later return from subsequent phases to Phase I. In this case the job is also rejected if $M > M_{\text{max}}$ or if the algorithm selects the same resource set $\mathcal{R}_{\text{job}}$ again.

3.2 Phase II — Examination of the Resources

This phase checks whether the initial set of resources provides a sufficient amount of resource properties, regardless of distribution.

As shown in Phase II, for each quantitative resource property P_i, the resources are sorted in descending order and the first N resources are checked to see whether they fulfil the given requirement d_i. If a check for any resource property fails, the algorithm returns to Phase I with M doubled. Otherwise, the algorithm proceeds to Phase III.

3.3 Phase III — Finding a Set of Potential Start Times

In this phase the potential start times are obtained from the timetables. As shown in Phase III, the algorithm first computes functions $a_{ij}(t)$, $A_j(t)$, and $A_j^s(t)$ for each resource R_j. The function $a_{ij}(t)$ expresses the amount of a resource property $P_i \in \mathcal{P}$ of a resource $R_j \in \mathcal{R}_{\text{job}}$ in time t. The function $A_j(t)$ indicates whether in time t the available amounts of resource properties of the resource R_j are sufficient for the request. The function $A_j^s(t)$ equals 1 if and only if the resource R_j is continuously available in the time interval $[t, t + t_{\text{res}}]$, or 0 otherwise.

Phase III: Finding a set of potential start times

foreach $R_j \in \mathcal{R}_{\mathrm{job}}$ **do**

$$\forall P_i \in \mathcal{P} \colon a_{ij}(t) = \begin{cases} p_{ij} - T_{ij}(t) & t \in [t_e, t_l + t_{\mathrm{res}}] \\ 0 & \text{otherwise} \end{cases}$$

$$A_j(t) = \begin{cases} 1 & \forall i \colon e_i \le a_{ij}(t) \\ 0 & \text{otherwise} \end{cases}$$

$$A_j^s(t) = \min_{t' \in [t, t+t_{\mathrm{res}}]} A_j(t')$$

end

$$\mathcal{T} = \{\, t \;;\; \exists R_j \in \mathcal{R}_{\mathrm{job}}, P_i \in \mathcal{P} \colon A_j^s(t) = 1 \wedge a_{ij}(t) \text{ increases}\}$$

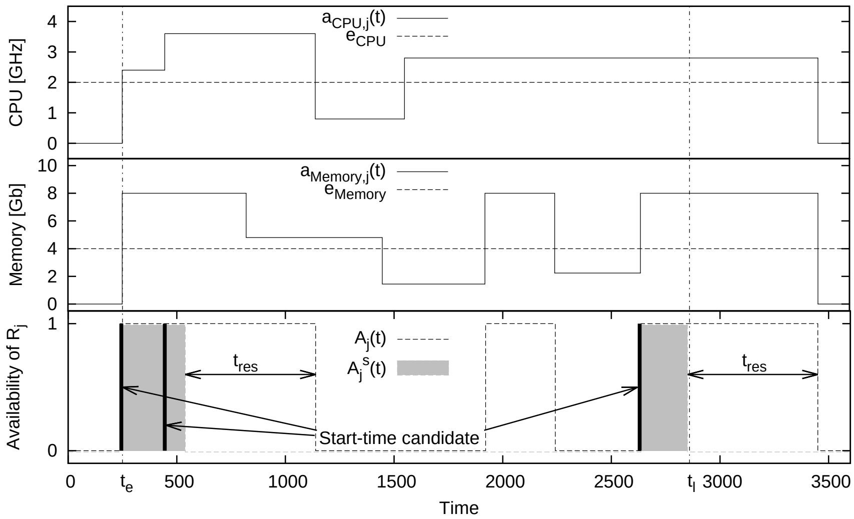

Figure 1: Construction of start-time candidates in Phase III – start time filtering.

Using these functions a set $\mathcal{T}$ containing all possible *start-time candidates* is constructed and passed to Phase IV as shown in Figure 1. The job can be started at any time within the grey interval, but only times in which the amount of a resource property of any resource increases must be considered. Note that any increase of a resource property at any node can result in the fulfilment of requirements d_i.

3.4 Phase IV — Construction of an Admissible Solution

In Phase IV the algorithm constructs an admissible solution based on the earliest possible start time.

For each $t \in \mathcal{T}$, from the earliest to the latest, it first computes the minimal amounts $a^t_{ij} = p_{ij} - T^t_{ij}$ of resource properties available during the whole reservation interval $[t, t + t_{\text{res}}]$ where $T^t_{ij} = \max_{t' \in [t, t+t_{\text{res}}]} T_{ij}(t')$. It then constructs a set $\mathcal{R}_a$ of available resources and tries to find a set $\mathcal{R}_s$ of N resources forming an admissible solution. If more than N resources are available, the set $\mathcal{R}_s$ is constructed by the `SelectAndSwap` function Phase IV. If the set $\mathcal{R}_s$ is admissible, the algorithm proceeds to Phase V; otherwise it returns to Phase I with M doubled.

The `SelectAndSwap` function only returns a non-empty set $\mathcal{R}_s$ of resources when the condition $u_i(\mathcal{R}_s, t') \leq 1$, $t' \in [t, t + t_{\text{res}}]$ is valid for all resource properties P_i after the job is placed. To simplify the computation of the condition, only the worst-case utilisation

$$\widetilde{u}_i(\mathcal{R}, t) = \frac{d_i + \sum_{R_j \in \mathcal{R}} T^t_{ij}}{\sum_{R_j \in \mathcal{R}} p_{ij}} \tag{3}$$

is considered. The `SelectAndSwap` function first tries to form an admissible resource set $\mathcal{R}_s$ by selecting N resources with high amounts of available resource properties. It cyclically sweeps through the resource properties and in each cycle selects the resource from $\mathcal{R}_a$ with the highest amount of the current resource property (lines 3 to 7). While $\mathcal{R}_s$ is not admissible, the algorithm tries to improve it by swapping the resources from $\mathcal{R}_s$ with those from $\mathcal{R}_a$ (lines 9 to 22). In each iteration it determines the resource property P_v for which the resources from $\mathcal{R}_s$ are most severely violate the utilisation. The function prefers to select $R_m \in \mathcal{R}_a$ with a high amount of resource property P_v, while it selects the resource from $\mathcal{R}_s$ at random. Resources are only swapped if no unfulfilled resource property gets worse and no previously fulfilled resource property becomes unfulfilled. The function gives up the search for an admissible solution when all resources from $\mathcal{R}_a$ have been tested for the same P_v or after $L_1 \cdot N$ attempts, where L_1 is a constant set by a system administrator.

Random swapping provides a wider sweep across multiple resource properties while keeping the algorithm simple. Furthermore, the randomness prevents the algorithm from getting stuck (as occurs, for example, when swapping of the same two resources in two consecutive iterations results in repetitive switching between two inadmissible solutions).

3.5 Phase V — Solution Optimisation

In Phase V the algorithm tries to improve the admissible solution by increasing the utilisation factor while retaining the same start time. As we are interested in the utilisation of the grid after the job is placed, the estimation $\widetilde{U}(\mathcal{R}, t) = \prod_{P_i \in \mathcal{P}} \widetilde{u}_i(\mathcal{R}, t)$ of the utilisation factor is used as a criterion.

Phase IV: Construction of an admissible solution

for $t \in \mathcal{T}$ *ordered in the ascending order* **do**
  $\mathcal{R}_a = \{R_j \in \mathcal{R}_{\text{job}}; A_j^s(t) = 1\}$
  $\forall R_j \in \mathcal{R}_a, \forall P_i \in \mathcal{P}$: $a_{ij}^t = p_{ij} - T_{ij}^t$
  switch $|\mathcal{R}_a|$ **do**
    case $< N$: $\mathcal{R}_s = \emptyset$
    case $= N$: $\mathcal{R}_s = \mathcal{R}_a$
    case $> N$: $\mathcal{R}_s =$ `SelectAndSwap(`$\mathcal{R}_a$`)`
  end
  if $\forall \mathcal{P}_i \in \mathcal{P} : \sum_{R_j \in \mathcal{R}_s} a_{ij}^t \geq d_i$ **then**
    go to Phase V
  end
end
$M = 2M$; go to Phase I

---

function $\mathcal{R}_s =$ `SelectAndSwap(`$\mathcal{R}_a$`)`
$\mathcal{R}_s = \emptyset$;
for $k = 1$ **to** N **do**
  $k' = (k-1) \bmod |\mathcal{P}| + 1$
  select (randomly) R_k from $\{R_j \in \mathcal{R}_a; \forall R_{j'} \in \mathcal{R}_a : a_{k'j}^t \geq a_{k'j'}^t\}$
  $\mathcal{R}_s = \mathcal{R}_s \cup \{R_k\}$; $\mathcal{R}_a = \mathcal{R}_a \setminus \{R_k\}$
end
$c = 1$; $m = 0$
while $(\exists P_i \in \mathcal{P} : \widetilde{u}_i(\mathcal{R}_s, t) > 1) \wedge (c \leq NL_1) \wedge (m \leq |\mathcal{R}_a|)$ **do**
  $m = m + 1$; $k = 1$
  let P_v be a random resource property from $\{P_i \in \mathcal{P}; \forall P_{i'} \in \mathcal{P} : \widetilde{u}_i(\mathcal{R}_s, t) \geq \widetilde{u}_{i'}(\mathcal{R}_s, t)\}$
  let $R_m \in \mathcal{R}_a$ be the m-th resource of $R_{\omega(1)}, R_{\omega(2)}, \ldots, R_{\omega(|\mathcal{R}_a|)}$ sorted by P_v in descending order
  let $R_{\pi(1)}, R_{\pi(2)}, \ldots, R_{\pi(N)}$ be resources of $\mathcal{R}_s$ in a random order
  repeat
    $\mathcal{R}_s' = \mathcal{R}_s \cup \{R_m\} \setminus \{R_{\pi(k)}\}$
    $canswap = \bigwedge_{i=1}^{|\mathcal{P}|} \left(\sum_{R_j \in \mathcal{R}_s'} a_{ij}^t \geq \min(\sum_{R_j \in \mathcal{R}_s} a_{ij}^t, d_i) \right)$
    if $canswap$ **then**
      $\mathcal{R}_s = \mathcal{R}_s'$; $\mathcal{R}_a = \mathcal{R}_a \cup \{R_{\pi(k)}\} \setminus \{R_m\}$; $m = 0$
    end
    $k = k + 1$; $c = c + 1$
  until $(k \leq N) \wedge (c \leq NL_1) \wedge (\neg canswap)$;
end

Phase V: Solution optimisation

if $|\mathcal{R}_a| = 0$ **then** go to Phase VI;
for $c = 1$ **to** NL_2 **do**
  let R_s be a random resource from $\mathcal{R}_s$
  let R_a be a random resource from $\mathcal{R}_a$
  $\mathcal{R}'_s = \mathcal{R}_s \cup \{R_a\} \setminus \{R_s\}$
  if $\big(\forall P_i \in \mathcal{P} : \widetilde{u}_i(\mathcal{R}'_s, t) \leq 1\big) \wedge \big(\widetilde{U}(\mathcal{R}'_s, t) \geq \widetilde{U}(\mathcal{R}_s, t)\big)$ **then**
    $\mathcal{R}_s = \mathcal{R}'_s$
    $\mathcal{R}_a = \mathcal{R}_a \cup \{R_s\} \setminus \{R_a\}$
  end
end

The algorithm performs $N \cdot L_2$ attempts to improve the admissible solution, where L_2 is a constant set by a system administrator. In each attempt it randomly selects one resource from the admissible solution and one from the other available resources. The resources are swapped if the new resource set forms an admissible solution with an increased utilisation factor.

3.6 Phase VI — Resource Reservation

Finally, the algorithm reserves the resources constituting the solution, updates the timetables, and returns the start time and the list of resources.

To reserve a quantitative resource property P_i on each resource in $\mathcal{R}_s$, the algorithm updates the timetables $T_{ij}(t') \leftarrow T_{ij}(t') + e_i + r_i \cdot (d_i - Ne_i)$, where $t' \in [t, t + t_{\text{res}}]$ is the reservation time interval and $r_i = (a^t_{ij} - e_i) / \sum_{j=1}^{N} (a^t_{ij} - e_i)$.

The reservation might fail because some of the selected resources have been reserved in the meantime by another co-allocation service. In this case the algorithm returns to Phase I without modifying the parameter M.

4 Experimental Work

The algorithm was implemented in the XtreemOS operating system and evaluated on the Grid'5000 infrastructure in terms of solution quality, time complexity, and operating system and infrastructure overheads.

4.1 Implementation

XtreemOS [Cortes et al. 2008, Nou et al. 2010] was used because of its modular architecture, which allows us to easily incorporate the presented co-allocation algorithm into the system. It also supports partial resource reservations.

Resource property (P_i)	Average ($\bar{p}_i$)	Standard deviation (σ_i)	Min	Max
Number of CPU cores	2.90	1.51	1	8
CPU frequency [GHz]	2.68	0.36	1.40	3.60
Total amount of RAM [GB]	2.95	0.84	0.73	7.71

Table 1: PlanetLab resource-property statistics for 606 nodes

The XtreemOS resource management system consists of the Service/Resource Discovery Service and the Application Execution Manager. It was built to support dynamic and heterogeneous grids where nodes can join and leave at any time, and may consist of a mix of cluster and desktop nodes. The core services run on dedicated nodes which can vary in number depending on the size of the system. This approach allows the resource management system to better adapt to changes in the grid and solve scaling and load-balancing problems [Foster and Jennings 2004]. Service/Resource Discovery Service agents work in a peer-to-peer fashion using a distributed hash table and provide resource information to the Application Execution Manager. The co-allocation service was integrated into the Application Execution Manager, which is responsible for managing jobs and resources.

4.2 Test Data

Grid'5000 does not support partial reservations and workload is represented by the list of accepted reservations. We therefore decided to use PlanetLab workload data instead, since these are measured and logged every five minutes. The PlanetLab platform [Chun et al. 2003] is a research environment consisting of diverse computer nodes, where each node can be used simultaneously by multiple users. For the experiments we selected 606 PlanetLab nodes for which data such as CPU frequency, number of CPU cores and memory were available. The experiments are based on platform-usage data for the working day 18 August 2010, midnight-to-midnight. The basic statistics of the nodes are presented in Table 1. The test cases were constructed from these data.

Each test case consists of a set of PlanetLab nodes already occupied according to the given platform-usage data and a new request that must be fulfilled by resources from this set. The same conditions were established on XtreemOS deployed on the Grid'5000 infrastructure.

The duration of the new request t_{res} was chosen from the set $\{10\text{ min},\ 30\text{ min},\ 1\text{ h},\ 2\text{ h},\ 3\text{ h},\ 5\text{ h}\}$ and the number of requested resources N from the set $\{2, 5, 10, 15, 30, 60\}$. The earliest start time t_e and the latest finish time $t_l + t_{\text{res}}$ were selected randomly from the intervals $[0\text{ h},\ 5\text{ h }50\text{ min}]$ and $[18\text{ h }10\text{ min},$

24 h], respectively. The value d_i of the resource property P_i in the new request was chosen randomly from the interval

$$[N(\bar{p}_i - 2\sigma_i),\, N(\bar{p}_i + 0.5\sigma_i)]\,, \tag{4}$$

where $\bar{p}_i$ and σ_i are the average and the standard deviation of p_{ij} over all nodes, respectively. By setting the upper boundary of the interval at an above-average level, we permitted some requirements to be difficult to fulfil. AMD Opteron nodes running XtreemOS were required by specifying qualitative resource properties.

For each number of requested resources we generated 90 test cases with simple requests and 90 test cases with collective requests sharing the same requirements except for e_i. For simple requests we set $e_i = d_i/N$ to require uniform distribution of resource properties, while for collective requests we set $e_i = d_i/2N$ to limit unequal distribution of resource properties.

4.3 Algorithm Parameters

The administrator can regulate the quality of solutions and the responsiveness of the system using the algorithm parameters M_{init}, M_{max}, L_1, and L_2, described in Section 3. Parameters M_{init} and M_{max} limit the initial and the final size of the search set. The influence of parameter L_1 is only expressed in critical cases, when it is hard to find the appropriate list of resources due to the highly utilised grid system or a complex request. The parameter L_2 regulates optimisation — larger values allow the algorithm to spend more time optimising the admissible solution computed in Phase IV at the expense of search time.

We set the parameters according to statistical data from the Nancy site of Grid'5000 [Orgerie and Lefèvre 2010]. The average request on the site required around 15 resources and had more than 25,000 reservations in its peak week. On average this means one successful request every 24 seconds, but during the busiest time of the day we can expect a new request every second.

We used a set of test cases requesting 15 resources to set M_{init}. To minimise the effect of operating system interactions, we used a single pass of the algorithm. We evaluated M_{init} for values 2, 3, 4, 6, and 8. The results show that $M_{\text{init}} = 3$ gives a good trade-off between solution quality and speed, as it provides 50% more solutions than $M_{\text{init}} = 2$ and only 7% less than $M_{\text{init}} = 4$, with $M_{\text{init}} = 4$ taking 60% more time in test cases with no solution. With values of $M_{\text{init}} > 4$ the increase of the number of solutions does not justify the increase in time consumption.

The algorithm consists of six phases, with the first and the last phase interacting with other operating system modules. To eliminate the influence of communication and infrastructure overheads in the experiments, only a single pass of Phases II to V was considered, i.e., $M_{\text{max}} = M_{\text{init}}$.

Table 2: Comparison of collective and simple approaches in all 540 test cases.

	Requested resources						
Solutions found with:	2	5	10	15	30	60	Σ
Simple approach	28	27	24	19	27	21	146
Collective approach	49	59	56	55	54	49	322
Earlier start with collective approach	4	7	1	7	5	2	26
Equal start time	24	20	23	12	22	19	120
Better utilisation (simple approach)	0	2	4	5	6	4	21
Better utilisation (collective approach)	7	13	15	7	16	15	73

The parameters L_1 and L_2 linearly increase the time consumption of Phases IV and V respectively (see Phase IV and V), while increasing the likelihood of finding the earliest solution with a high utilisation factor. We tested parameters L_1 and L_2 in a range from 2 to 50 and found that neither the number nor the quality of the admissible solutions obtained improved significantly after $L_1 = 7$ and $L_2 = 10$.

With parameters $M_{\text{init}} = 3$, $L_1 = 7$, and $L_2 = 10$ the algorithm search time for a 15 resource request is less than 0.1 s and for a 60 resource request less than 1 s on average on a 2.2 GHz CPU AMD Opteron 275. This ensures a good system response at peak request times and a high probability of successful reservation in Phase VI. The chosen parameter values were used in the experiments.

4.4 Quality of Solution

To assess the performance of the described algorithm, its outputs were compared with solutions obtained by specifying simple requests. Due to the stochastic nature of the algorithm, the search in each test case was repeated 1000 times.

The results in Table 2 show that the simple approach provided a solution in 27% of the test cases, while the collective approach is better since it provided a solution in 60% of the test cases. The collective approach started jobs earlier in 18% of the test cases in which both approaches obtained a solution. More specifically, the results showed that jobs are started on average 3 h earlier. Furthermore, in test cases where both approaches obtained solutions with the same start times, the collective approach provided more than three times as many test cases with a better utilisation factor than the simple approach. Detailed analysis showed that the relative improvement of the utilisation factor also favours the collective approach, by 8.5% on average. It is important to note that the results show no correlation between the number of requested nodes and the probability of finding a solution.

We also compared the results of the simple and collective approaches to op-

Table 3: Comparison of collective and simple approaches and optimal solutions for 5 resources.

	Collective	Simple
Solution found	59	27
Suboptimal (later) start time/average delay [h]	3/1.5	7/2.5
Utilisation factor at same start times (portion of the optimal utilisation factor)	0.92	0.84
Test cases where utilisation factor is above 99% of the optimal	22	4

timal solutions. Given the NP-completeness of the problem, a brute-force search for the optimal solution is extremely time-demanding. To make the process feasible, we limited the analysis to the test cases requesting five resources.

A brute-force search finds a solution in 59 test cases. Table 3 shows that the collective approach is successful in all of these test cases. Only three of the solutions have suboptimal start times, while others have an average utilisation factor 92% of the optimal. Detailed analysis showed that the collective approach found 22 test cases with a utilisation factor higher than 99% of the optimal. Overall, the results obtained by collective approach are clearly better than those obtained by the simple approach.

4.5 Time Complexity

The search time depends on grid utilisation, the number of requested resources and request complexity. The average search times presented in Figure 2 increase with the number of requested resources. The simple approach is faster, especially in test cases without a solution. In these cases it gives up the search in the early stages because of the high values of individual requirements preventing the algorithm from collecting enough resources to form a solution.

The collective approach requires more time than the simple approach. The search time of the collective approach in test cases without a solution is mainly attributed to additional sweeps for possible start times in Phase IV. Search times increase as the number of requested resources grows. Phase II reduces the average search times of test cases without a solution. In test cases requesting 5 or 10 resources Phase II rejected about half of the test cases without a solution.

The relative time consumption of Phase II is less than 1%, while Phase III needs around 5% of the total search time. The rest of the time is consumed by Phases IV and V. If Phase IV is reached and if an admissible solution is hard to obtain, Phase IV takes more time than Phase V. Otherwise the majority of the search time is consumed by Phase V. The average improvement of the utilisation

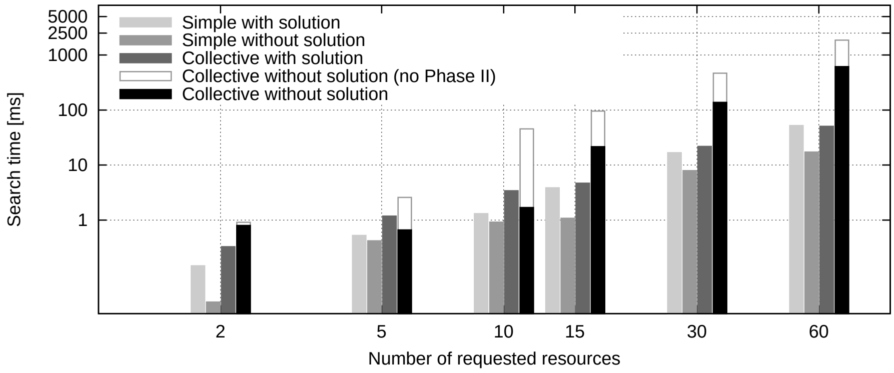

Figure 2: Average algorithm search times as a function of the number of requested resources.

Table 4: Search time statistic of 90 test cases requesting 5 nodes.

	Phases I-VI	Phases II-V
Minimal [ms]	126.4	0.008
Maximal [ms]	324.9	9.167
Average [ms]	169.8	1.004
Standard deviation [ms]	29.4	1.664

factor in Phase V is about 10%.

4.6 Operating System and Infrastructure Overheads

To assess the total time consumption of the algorithm (including Phases I and VI), we deployed the XtreemOS operating system on eight 2.6 GHz AMD Opteron 2218 and seven 2.2 GHz AMD Opteron 275 Grid'5000 nodes. The Application Execution Manager with our co-allocation module ran on one of the 2.2 GHz AMD Opteron 275 nodes. We preset the resource properties and the reservations of each XtreemOS resource to reflect the same state of the resources described in the test cases. The experiment was limited to test cases requesting 5 resources. After the algorithm performed the co-allocation, the timetables and resource properties were reset to enable new independent tests. Each test case was repeated 100 times.

The time consumption of XtreemOS is presented in Table 4. The overhead of Phases I and VI, which amounts to almost 170 ms on average, is attributed to network communication, request message processing and actions of other XtreemOS modules such as user authentication, monitoring and logging.

5 Conclusion

The most common co-allocators today lack the ability to adapt to a heterogeneous and dynamic grid infrastructure. The co-allocation algorithm described in this paper presents a novel approach to tackling these issues.

The main idea of the proposed co-allocation algorithm is to extend simple co-allocation requests with collective requirements, support for partial resource reservation, and optimisation of job placement in terms of grid utilisation. The algorithm uses a heuristic approach and consists of six phases: it limits the search to a reasonably sized resource set, checks the obtained resource set timetables, selects possible start times, searches for admissible solutions and tries to find the most appropriate, resource set. The algorithm cannot guarantee the optimal solution, because the search may last longer than the user is willing to wait. To obtain a compromise between the quality of the solution and the search time, the search stops after a given number of iterations.

We implemented the proposed algorithm in the XtreemOS grid operating system. The co-allocation module of any other operating system can use this algorithm to support collective requests, although partial resource reservation can only be used if the grid system supports this functionality.

The experiments on the Grid'5000 infrastructure clearly show that, despite its simplicity, in most cases the algorithm finds an appropriate solution if one exists. The algorithm finds twice as many solutions if collective requests are used instead of simple requests. It thus contributes to better utilisation of the system. The search time of the algorithm is dependent above all on the number of requested nodes and current grid utilisation.

Each time it runs, the algorithm performs optimisation of grid utilisation on a subset of all grid nodes. Since this subset change every time the algorithm is invoked, local optimisation hopefully leads to global optimisation of grid usage.

In the future we intend to explore grid-usage tendencies to predict the probability of the existence of a solution for a given co-allocation request based on the current state of a grid. By means of grid-usage evaluation, we plan to improve the performance of the search. In addition, by including the possibility of rescheduling existing reservations, it would be possible to achieve even better utilisation and efficiency, as well as better management of resource failures.

Acknowledgements

This research was funded partly by the European Union (European Social Fund) and partly by the European FP6 project XtreemOS under EC contract number IST-033576. The experiments presented in this paper were carried out using the Grid'5000 experimental test bed, developed under the INRIA ALADDIN development action, with support from CNRS, RENATER and several universities, as well as other funding bodies (see https://www.grid5000.fr).

References

[Cankar et al. 2010] Cankar M., Hadalin P., and Artač M. "Co-allocation of computational resources in XtreemOS grids"; Proc. ERK'10, IEEE (2010), 10–13.

[Cappelo et al. 2005] Cappello F., Caron E., Dayde M., Desprez F., Jegou Y., Primet P., Jeannot E., Lanteri S., Leduc J., Melab N.: "Grid'5000: a large scale and highly reconfigurable grid experimental testbed",GridComputing Workshop IEEE, (2005) 99–106.

[Castillo et al. 2009] Castillo C, Rouskas G, Harfoush K.: "Resource co-allocation for large-scale distributed environments"; Proc. HPDC'09, ACM, (2009), 131–140.

[Chun et al. 2003] Chun B., Culler D., Roscoe T., Bavier A., Petterson L., Wawrzoniak M., Bowman M.: "Planetlab: An Overlay Testbed for Broad-Coverage Services"; Proc. ACM SIGCOMM 33, 3, (2003), 3–12.

[Cortes et al. 2008] Cortes T, Franke C, Jégou Y, Kielmann T, Laforenza D.: "XtreemOS: a vision for a Grid operating system"; White paper, (2008);

[Czajkowski et al. 1999] Czajkowski K., Foster I., Kesselman C.: "Resource coallocation in computational grids"; Proc. HPDC'99, IEEE, (1999) 219–228.

[Elmroth and Tordsson 2007] Elmroth E. and Tordsson J.: "A standards-based Grid resource brokering service supporting advance reservations, coallocation, and cross-Grid interoperability"; Concurrency and Computation: Practice & Experience, 21, 18, (2009), 2298–2335.

[Foster et al. 1999] Foster I, Kesselman C., Lee C., Lindell B, Nahrstedt K., and Roy A.: "A distributed resource management architecture that supports advance reservations and co-allocation"; Proc. IWQoS'99 IEEE, (1999), 27–36.

[Foster and Jennings 2004] Foster I. and Jennings NR.: "Brain meets brawn: Why grid and agents need each other"; Agents and Multiagent Systems, (2004), 8–15.

[García-Peñalvo et al. 2012] García-Peñalvo F.G., Forment M.A., and Lytras M., "Some Reflections about Service Oriented Architectures, Cloud Computing Applications, Services and Interoperability J. UCS Special Issue"; Journal of Universal Computer Science, 18, 11, (2012), 1405–1409.

[Liu and Foster 2003] Liu C, Foster I.: "A constraint language approach to grid resource selection"; Proc. HPDC-12 (2003), 1–13.

[MacLaren 2007] MacLaren J.: "HARC: the highly-available resource co-allocator"; Proc. 2007 OTM confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS-Volume Part II, (2007); 1385–1402.

[Mohamed and Epema 2008] Mohamed H, Epema D.: "KOALA: a coallocating grid scheduler"; Concurrency and Computation: Practice and experience 20, 16, (2008), 1851–1876.

[Netto and Buyya 2010] Netto MAS, Buyya R.: "Co-Resource Co-allocation in Grid Computing Environments"; Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications, chap. 20. 1 edn., Information Science Publishing - Imprint of: IGI Publishing: Hershey, PA, (2010), 476–494.

[Nicolas et al. 2011] Nicolas C., Joseph E., and Martin S.. "OAR Documentation - User Guide"; Technical report, also appeared as electronic version http://oar.imag.fr/documentation/.

[Nou et al. 2010] Nou R., Giralt J., Corbalan J., Tejedor E., Fitó J.O., Perez J.M., and Cortes T., :"Xtreemos application execution management: A scalable approach"; Proc. GRID'10, IEEE, vol. 2 (2010), 49–56.

[Orgerie and Lefèvre 2010] Orgerie, A. Lefèvre, L.: "A year in the life of a large scale experimental distributed system: the Grid'5000 platform in 2008"; Technical Report, INRIA, RR–7481, (2010).

[Takefusa et al. 2007] Takefusa A., Nakada H., Kudoh T., Tanaka Y., Sekiguchi S.: "GridARS: An advance reservation-based grid co-allocation framework for distributed computing and network resources"; Proc. JSSPP'07, Springer-Verlag, (2007), 152–168.

[Younge et al. 2011] Andrew J. Younge, Robert Henschel, James T. Brown, Gregor von Laszewski, Judy Qiu, and Geoffrey C. Fox: "Analysis of Virtualization Technologies for High Performance Computing Environments"; Proc. CC'11, IEEE, (2011), 9–16.

Appendix – NP-Completeness

Many scheduling problems are similar to the problem of co-allocation as defined in Section 2. The list of similar NP-complete problems includes minimum flow shop scheduling, minimum job shop scheduling, and minimum multiprocessor scheduling [Garey and Johnson 1979]. Due to the specific form of the criterion function, i.e. the multiplication of utilisations, we provide a proof that the presented co-allocation problem is NP-complete. The proof is made for a subproblem where all resources are free and $e_i = d_i/(2N)$. If these two constraints are removed, the problem does not become any simpler.

In the general case, we assume a set $\mathcal{R}$ of n resources and a set $\mathcal{P}$ of m quantitative resource properties, i.e., $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ and $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. The quantity of a resource property P_i provided by a resource R_j is denoted by p_{ij} and a required amount of a resource property P_i is denoted by d_i. Furthermore, we assume $p_{ij} \in \mathbb{N}_0$ and $d_i \in \mathbb{N}_0$.

A requirement d_i is fulfilled by a subset of resources if their combined quantity of the resource property P_i is greater than or equal to d_i. The goal is to find N resources which together fulfil the requirements for all resource properties.

However, the minimal amount of a resource property provided by a single resource should be not less than one half of the average provided by all the selected resources. To avoid selecting resources that provide too much of any particular resource property, the product of all the resource properties provided by all the resources should not exceed the product of all the requirements by more than a certain factor. The decision problem regarding the fulfilment requirements is formulated as follows:

Optimal Requirements Fulfilment
INSTANCE: $P \in \mathbb{N}_0^{m \times n}$ of p_{ij}'s and $D \in \mathbb{N}_0^n$ of d_i's; positive integers $N \leq n$, a and b where $a \leq b$.
QUESTION: Is there a set $\mathcal{S} \subseteq \{1, 2, \ldots, n\}$ such that $|\mathcal{S}| = N$ where

$$\sum_{j \in \mathcal{S}} p_{ij} \geq d_i \quad \text{and} \quad \forall j \in \mathcal{S} \colon p_{ij} \geq \frac{d_i}{2N} \quad , \tag{1}$$

for all $i \in \{1, 2, \ldots, m\}$, and

$$\prod_{i\colon d_i \neq 0} \left(\frac{d_i}{\sum_{j \in \mathcal{S}} p_{ij}} \right) \geq \frac{a}{b} \quad ? \tag{2}$$

Theorem 1. *The optimal requirements-fulfilment problem is NP-complete.*
PROOF: The optimal requirements-fulfilment problem clearly belongs to NP as it takes time $\mathcal{O}(n \cdot m)$ to verify a solution.

To prove that the optimal requirements-fulfilment problem is NP-complete, we reduce the minimum-cover problem, itself an NP-complete problem, to it. The minimum-cover problem is formulated as follows:

Minimum Cover (SP5 in [Garey and Johnson 1979])
INSTANCE: Collection $\mathcal{C}$ of subsets of a finite set U, i.e., $\mathcal{C} \subseteq 2^{\mathcal{U}}$; positive integer $k \leq |\mathcal{C}|$.
QUESTION: Does $\mathcal{C}$ contain a cover for $\mathcal{U}$ of size k or less, i.e. does there exist a collection $\mathcal{C}' \subseteq \mathcal{C}$ such that $\cup_{C \in \mathcal{C}'} C = \mathcal{U}$ and $|\mathcal{C}'| \leq k$?

To reduce the minimum-cover problem to the minimal requirements-fulfilment problem, we transform an instance $\langle \mathcal{U}, \mathcal{C}, k \rangle$ of the minimum-cover problem to k instances of the minimal requirements-fulfilment problem, one instance for every $k' \in \{1, 2, \ldots, k\}$ (note that $k \leq |\mathcal{C}|$ and so the value of k is polynomic in the size of the input).

To construct an instance of the minimal requirements-fulfilment problem for a particular k', we consider all the elements u_i of the set $\mathcal{U}$ as resource properties and all the sets $C_j \in \mathcal{C}$ as resources. Hence, $m = |\mathcal{U}|$ and $n = |\mathcal{C}|$. Furthermore, let p_{ij} be defined as

$$p_{ij} = \begin{cases} 1 & u_i \notin C_j \\ 2k' & u_i \in C_j \end{cases}$$

and let $d_i = 2k'$. Finally, let $a = 1$ and $b = k'^{|\mathcal{U}|}$. The transformation can be made in polynomial time: the collection C can be transformed in time $\mathcal{O}(m \cdot n)$ to yield P, and D can be generated in time $\mathcal{O}(n)$.

The instance of the minimal requirements-fulfilment problem for a particular k' has a solution if and only if $\mathcal{C}$ contains a cover for $\mathcal{U}$ of size k'. To see that, one should make two observations. First, because $\sum_{j \in \mathcal{S}'} p_{ij} \leq 2k'^2$ and $\prod_{i\colon d_i \neq 0} d_i = (2k')^m$, condition (2) is always fulfilled as

$$\prod_{i\colon d_i \neq 0} \left(\frac{d_i}{\sum_{j \in \mathcal{S}'} p_{ij}} \right) \geq \prod_{i\colon d_i \neq 0} \left(\frac{2k'}{2k'^2} \right) = \frac{1}{k'^m} \quad .$$

Second, if requirement d_i is fulfilled, then there exists at least one j so that $p_{ij} = 2k'$ and hence $u_i \in C_j$ for $C_j \in \mathcal{C}'$ (if such j does not exist, then $\sum_{j \in \mathcal{S}'} p_{ij} =$

$k' \leq 2k'$); if u_i is covered by some $\mathcal{C}'$, than there exists some j so that $p_{ij} = 2k'$ and $\sum_{j \in \mathcal{S}'} p_{ij} \geq 2k'$.

Finally, to solve an instance of the minimum-cover problem, at most k instances of the optimal requirements-fulfilment problem must be checked. □

Corollary 2. *The optimisation version of the optimal requirements-fulfilment problem is NP-hard.*

References

[Garey and Johnson 1979] Garey MR, Johnson DS.: "Computers and Intractability; A Guide to the Theory of NP-Completeness"; W. H. Freeman & Co./New York, USA, (1990).