Autonomous Resource-Aware Scheduling of Large-Scale Media Workflows

Stein Desmet, Bruno Volckaert, Filip Turck

Autonomous Resource-Aware Scheduling of Large-Scale Media Workflows

Stein Desmet, Bruno Volckaert, and Filip De Turck

Department of Information Technology(INTEC) - IBCN, Ghent University - IBBT,
Gaston Crommenlaan 8 bus 201, B-9050 Gent, Belgium,
stein.desmet@intec.ugent.be

Abstract. The media processing and distribution industry generally requires considerable resources to be able to execute the various tasks and workflows that constitute their business processes. The latter processes are often tied to critical constraints such as strict deadlines. A key issue herein is how to efficiently use the available computational, storage and network resources to be able to cope with the high work load. Optimizing resource usage is not only vital to scalability, but also to the level of QoS (e.g. responsiveness or prioritization) that can be provided. We designed an autonomous platform for scheduling and workflow-to-resource assignment, taking into account the different requirements and constraints. This paper presents the workflow scheduling algorithms, which consider the state and characteristics of the resources (computational, network and storage). The performance of these algorithms is presented in detail in the context of a European media processing and distribution use-case.

1 Introduction

In the audiovisual industry, the resources required to perform various tasks such as editing and processing of video and audio, are generally quite considerable, not to mention expensive. This is not only true for computational resources but for storage resource and communication resources as well. Converting video material from one format to another can easily take several hours, even on today's multi-core servers. Raw Standard Definition material used during the editing process has bitrates of 60Mbps or more, while HD material easily quadruples these bitrates. This obviously creates high demands on storage and network. Especially for independent film makers, these resource requirements and their cost might pose an obstacle.

The Independent Films In Progress (IFIP) [1] project designs a platform aimed at the media industry, especially the independent film. Its goal is to provide a framework supporting the development of independent productions. Amongst others, it provides a platform for users to automate and submit production processes. The platform offers the possibility for users to submit a series of tasks - e.g. video transcoding or CGI rendering - or *workflows* they want to be executed within a certain budget and timeframe. The platform autonomously locates appropriate resources for the execution of these workflows, and determines the most appropriate start time of the workflows, taking all their constraints

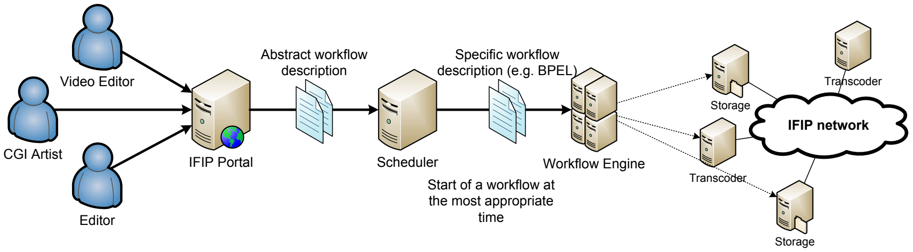

Video Editor
CGI Artist
Editor
IFIP Portal
Abstract workflow description
Scheduler
Specific workflow description (e.g. BPEL)
Workflow Engine
Start of a workflow at the most appropriate time
Storage
Transcoder
IFIP network
Transcoder
Storage

Fig. 1. Overview of the IFIP platform. The scheduler locates resources for the workflows submitted at the portal, and launches them at an appropriate time.

into account. This is illustrated in Figure 1. A user submits a workflow at the IFIP portal, which forwards the flow to the scheduler. The scheduler examines the workflow and the user's constraints defined in the metadata. For each task in the workflow, the scheduler locates resources hosting the required services or applications - taking into account both the user's demands and the current state of the infrastructure - and adds the task-to-resource mapping to the workflow description. Additionally, an optimal starttime for the workflow is determined as well. At that starttime, the scheduler submits the workflow description to the workflow engine, which subsequently executes the workflow.

Considering such a multi-client distributed media processing platform, an efficient allocation of all resources - computational, storage and communication - is of the utmost importance for being able to meet the users' requirements and offer high Quality-of-Service such as responsiveness or budget considerations. Efficient resource usage also results in the ability to handle higher loads. This optimal resource selection is not only location-based (which resource to use), but also time-based (when to use the resource). This scheduling problem is known to be NP-hard [2]. There is already a substantial amount of research on the topic of scheduling tasks in grids and other heterogeneous distributed environments [3–5].

However, media-oriented cases, including the one presented here, have specific, highly data-intensive and time-constrained demands, to which the workflow-to-resource mapping must be specifically tuned. The network in a media environment for example requires special consideration. Due to typical network packet patterns inherent to media transfers, media traffic turns out to behave differently from regular IT traffic. This makes unreliable throughput, packet loss and lost connections due to oversubscription far more likely than in normal situations [6, 7].

The MediaGrid framework (and accompanying MediaNSG simulator) [8] is aimed at employing grid technology in a media production and distribution environment. The IFIP project builds further on this framework, and the research work performed for the GEISHA project [7], the predecessor to the IFIP project. The GEISHA project focused on the actual implementation of a service oriented architecture in media grids, and the associated challenges.

The research project GridCast [9] is undertaken by the British Broadcasting Company (BBC) and the Belfast e-Science Center, and also aimed at developing a media grid. However, it is intended on the specific BBC topology, whereas we aim to provide a general framework. Furthermore, scheduling in GridCast is mainly focused on scheduling data transfers only.

Another project aimed at media oriented grids is MediaGrid.org [10], an open standards group for the development of a grid platform specifically designed for digital media. Other projects include mmGrid [11], a middleware for supporting multimedia applications in a grid environment and Parallel-Horus [12], a cluster programming library for applications in multimedia grids. However, these projects mainly focus on design and implementation of media grids, rather than the problem of making efficient use of available grid resources.

This paper presents a number of distinct algorithms to schedule workflows to resources in a media environment. The heuristics presented are an adaption of the list scheduling technique [13] and use an offline - sometimes also referred to as static - scheduling approach. This refers to the time at which scheduling decisions are taken. In the offline approach, information regarding the state of all resources and every workflow is assumed to be known. This is a valid approach, as users submit their workflows at the portal, and expect results by a certain time, allowing the scheduler to periodically perform scheduling of the pending workflows. For example, each workflow that needs to be executed the next day is known, so scheduling decisions can be made overnight. Online scheduling on the other hand schedules workflows 'on-the-fly' when they are submitted, based on the current system load. This increases responsiveness, but leads to less efficient resource usage, as the global overview of the system is reduced.

This article is organized as follows. Section 2 describes the problem in greater detail and defines the model used. Section 3 gives an overview of the designed algorithms, while Section 4 thoroughly evaluates these algorithms. Finally, Section 5 presents directions for future research.

2 Problem Model Description

We assume k workflows $\{w_1, w_2, \ldots, w_k\}$ competing for resources, whose submit time and deadline are known as well as their maximum allowed budget. Each workflow consists of a set of inter-dependent atomic tasks $\{t_1, t_2, \ldots, t_n\}$ which are represented in a Directed Acyclic Graph (DAG). The edges between nodes represent task dependencies, and a node either represents a task or a control structure such as a decision or a parallel construct. The particular execution path that will be chosen by a decision construct is not known until runtime, i.e. workflows are non-deterministic. By nature, a loop construct is not supported by a DAG. Loops can be handled by for example unfolding or peeling them as described in [14].

Each task requires a specific service to execute, such as for example a transcoder or a CGI renderer. It retrieves its input data from a data repository, and stores its output data on a data repository. There is no direct data exchange between tasks, unless through an intermediary data repository. Tasks can retrieve and store their data either streaming or non-streaming. Each task requires a predefined processing time on a standard processor.

The network consists of a set of nodes, interconnected by a number of edges. These edges represent network connections and are considered to be unidirectional. In other words, to create a full duplex link, two edges are needed. Links offer a certain bandwidth at a certain use cost. A node represents a network node, to which a Computational Resource (CR) or Data Resource (DR) can be attached. The routing between two nodes is known in advance and is assumed to be fixed.

Computational resources offer one or more service types - e.g. transcoding or CGI rendering - at a certain execution speed, expressed in relation to a standard processor. Data Resources are the data repositories and can be used for both data retrieval as for storing output data. Similar to Computational Resources, Data Resources have a particular read speed and write speed, and an associated use cost.

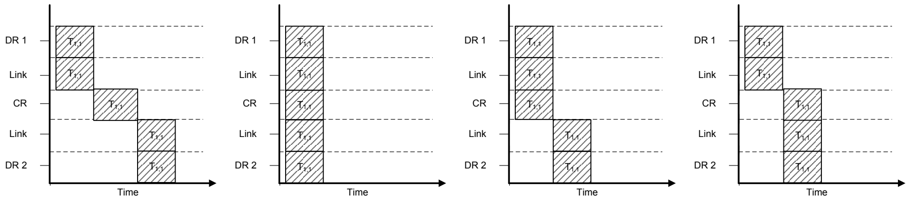
Every task must make *advance reservations* on each resource (computational, data, network) it uses. In other words, the task reserves exclusive use of the resource during a certain period. Note that this reservation system also helps to avoid the earlier mentioned network oversubscription danger. Rather than reserving each resource for the entire duration of a task, we have chosen to reserve them only when they are actually needed (i.e., a Data Resource used for data retrieval is only reserved for the time it takes to actually read the input, not for the whole input - processing - output cycle). This improves resource usage, but complicates the reservation procedure, as dependencies between various reservations need to be taken into account. Furthermore, for Data Resources a distinction is made between reservations for reading and writing. It is assumed that existing files are never overwritten, so reading and writing from the same Data Resource can occur simultaneously.

Furthermore, it is required that every resource can throttle its speed. Suppose a Computational Resource is able to perform a task at a certain speed, but the Data Resource is not able to supply the input data at that speed. It is then necessary for the Computational Resource to slow down until both speeds match.

3 Scheduling Algorithm Details

The scheduling algorithm's aim is to find suitable resource triplets (i.e., input Data Resource, Computational Resource and output Data Resource) with sufficient network interconnections for each workflow activity. Resource interconnection characteristics are equally responsible for workflow execution times (and cost) and must be taken into account, in order to avoid creating network bottlenecks.

Determining a suitable set of reservations for a resource combination is no trivial task. Reservation length depends on resource speed and task requirements, but also on whether or not the task is streaming its data. Streaming input requires equal duration of input retrieval and task execution, as we assume input data will be streamed during the full time spent on processing. As



(a) No streaming (b) Input and output streaming (c) Input streaming (d) Output streaming

Fig. 2. Resource reservation order according to streaming or non-streaming nature.

mentioned earlier, this may require throttling to match both resources' speeds, thus ultimately influencing reservation lengths. This obviously creates dependencies between the reservations of a single task. Furthermore, reservations need a specific order, depending on the nature of the input and output, as illustrated in Figure 2. The figure shows the different reservation orders that must be respected when dealing with streaming/non-streaming input/output data of individual workflow activities.

Apart from these task-level reservation restrictions, there are also workflow-level restrictions on reservations. Tasks can only execute when all parent tasks have finished. Overlap of reservations on the same resource is not allowed, but provisions are made to support workflows with alternate execution paths. As the actual execution path is not known until runtime, reservations of tasks in alternate paths are allowed to overlap. Only one set of reservations will effectively be used at runtime.

The algorithms presented here are based on the List Scheduling technique [13]. The general outline is shown in Listing 3.1. The heuristic is initialized by adding the root nodes of every workflow to the list Q. Subsequently, a priority is assigned to each task in Q according to a certain strategy, discussed in Section 3.1. The task with the highest priority is selected for assignment and removed from Q. The list Q is somewhat similar to a priority queue, but the priorities of its content need to be re-evaluated in each iteration of the algorithm, as the priority of an unassigned task is usually based on that task's properties, the tasks that have already been assigned, and the associated system loads. The latter two change after every iteration of the algorithm.

The selected task is then assigned to a triplet of resources and network interconnections according to certain criteria. Suitable reservations need to be made on the selected resources, taking into account existing reservations and inter-reservation dependencies. Different approaches can be considered to select the resources and reservations, presented in Section 3.2.

If no acceptable resource combination can be found for the selected task because each possible combination would violate a constraint, action is undertaken to handle this situation. As with deploying tasks and assigning scores, multiple approaches can be considered (Section 3.3).

Algorithm 3.1: SOLVE($Workflows, Resources$)

```
$Q, Processed \leftarrow [\,]$
for each $w_i \in Workflows$
  do $Q \leftarrow Q + root(w_i)$
while $Q \neq [\,]$
  do
    $HighestPriority \leftarrow 0$
    $CurrentTask \leftarrow null$
    for each $t_j \in Q$
      do
        $Priority \leftarrow$ GETPRIORITY($t_j$)
        if $Priority > HighestPriority$
          then
            $HighestPriority \leftarrow Priority$
            $CurrentTask \leftarrow t_j$
    $Q \leftarrow Q - CurrentTask$
    $Result \leftarrow$ ASSIGN($CurrentTask, Resources$)
    if $Result = Constraint\_Violation$
      then HANDLECONSTRAINTVIOLATION($CurrentTask, Q, Resources$)
      else
        $Processed \leftarrow Processed + CurrentTask$
        for each $t_j \in children(CurrentTask)$
          do
            if $\forall t_k \in parents(t_j) : t_j \in Processed$
              then $Q \leftarrow Q + t_j$
```

Finally, the children of the selected task are considered for insertion in Q, but only if all of a child's parent tasks have already been assigned to resources. This is necessary to preserve the execution order of the workflow.

This entire process is repeated until there are no more tasks left in Q.

In this discussion, we focus on two optimization objectives, the makespan of a workflow - its entire execution length - and the execution cost - the total cost for using the resources. Task deadlines and budget requirements are considered to be strict i.e. violating them means the workflow cannot be executed.

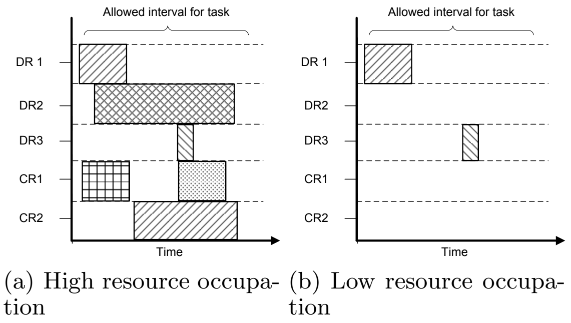
3.1 Determining task importance

The following strategies to assign priorities to tasks in Q have been designed.

Random (RC) Assigns random priorities to tasks.

ClosestConstraintViolation (CCV) This algorithm assigns higher priority to tasks closer to violating a workflow constraint, i.e. are closer to their deadline or budget. This naturally depends on the tasks of the workflow that have already been assigned. Giving priority to workflows closer to violating a constraint may prevent these workflows from violating that constraint, as they gain precedence for using the available resources.

ClosestRelativeConstraintViolation (CRCV) This algorithm is similar to the previous one, but the time or budget remaining values are first normalized, and priority is given based on these normalized values. In other words, a task whose workflow already uses a high *percentage* of its allowed budget or time interval, receives priority.



(a) High resource occupation (b) Low resource occupation

Fig. 3. Two different tasks and the current reservations on their available resources

TaskLeastAvailableResourceTime (TLART) This approach considers the set of available resources (computational, data and network) for a task and the occupation on these resources. This occupation is evaluated during the time interval in which the task is allowed to execute. Tasks where the average occupation across their available resources during this interval is higher, receive higher priority. In other words, tasks whose available resources are already heavily in use receive priority. Figure 3 shows the available resources and their reservations for two different tasks. The resources of the task in Figure 3(a) are obviously more occupied than the resources of the task in 3(b). TLART would consequently give higher priority to the former task.

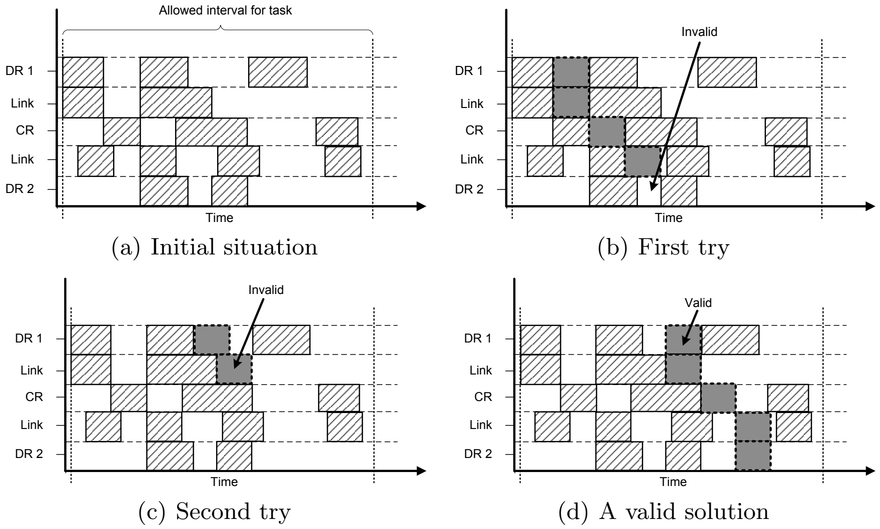
TaskMostAvailableResourceTime (TMART) This heuristic is exactly the opposite of TLART, as it gives precedence to tasks with lower average resource occupations. In Figure 3, TMART gives higher priority to the task in Figure 3(b). This is a somewhat greedy strategy, as tasks that are already known to have a high probability of finding a suitable assignment are given priority.

3.2 Assigning tasks to resources

The following strategies are available to assign tasks to resources.

RandomInserter (RI) Randomly chooses a Computational Resource able to process the task and an input and output Data Resource containing the required data sets. The task is scheduled to start as early as possible, taking into consideration the current reservations on the Computational, Data and network resources.

This process of finding reservations for a task is illustrated in Figure 4 for a task with non-streaming input and output. Figure 4(a) shows the initial situation. Figure 4(b) shows a first attempt at finding reservations for the current task. The first free interval on the input Data Resource is valid and an overlapping free interval exists for the network link from the Data Resource to the Computational Resource. An adjacent interval is available on the Computational Resource and another adjacent interval is present on the network link going from



(a) Initial situation (b) First try

(c) Second try (d) A valid solution

Fig. 4. Finding the earliest possible reservations and starttime for a task

the Computational Resource to the output Data Resource. However, there is no valid interval available on the output Data Resource. Figure 4(c) shows the second attempt at finding suitable reservations. The second free interval on the input Data Resource is again valid, but if reading from the resource is started as early as possible, then there is insufficient overlap with the free interval on the network link. The correct solution is shown in Figure 4(d). Figure 4 only shows the case of a task with non-streaming input and output. The three other cases are handled differently, as input-, output- and processing time depend on each other.

SimpleInserterWithDelay (SIWD) The algorithm starts by choosing a Computational Resource from the available resources, and subsequently proceeds by choosing an input and output Data Resource for that particular Computational Resource. The resource is selected in terms of completion time, lowest use cost or a weighted average. Determining the best resource in terms of completion time considers both a resource's speed and occupation, by looking for the earliest usable time slot on that resource as shown in Figure 5. In other words, a slower resource might be picked over a faster resource because it has an earlier time slot available. Reservations are made as described earlier for the random algorithm.

Note that this strategy considers each resource separately. A Computational Resource might be selected because it has a very early available time slot, but this slot is not usable due to reservation dependencies on the other resources, and the Computational Resource may have been a poor choice. Furthermore, the practice of first choosing a Computational Resource and then Data Resources can often lead to suboptimal solutions [15].

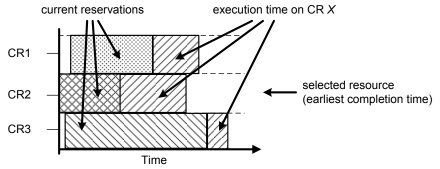
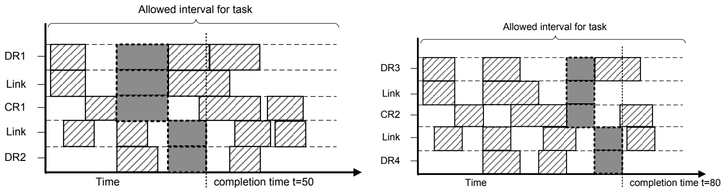


Fig. 5. Resource selection by the SimpleInserterWithDelay strategy



(a) Resource combination 1 (b) Resource combination 2

Fig. 6. Resource selection by the BestCombinationInserter strategy

BestCombinationInserter (BCI) This strategy improves SIWD by evaluating every possible resource combination of input Data Resource, output Data Resource and Computational Resource - including the network links involved - according to the criteria earliest completion time, lowest use cost or a weighted average of both. To determine the earliest completion time of a task on a particular resource combination, the earliest possible valid set of resource reservations must be found on that combination. The procedure for finding this reservation set is similar to the reservation procedure described for the RandomInserter algorithm (Figure 4). The BCI strategy is obviously computationally considerable more expensive than SIWD.

BestCombinationInserter is illustrated in Figure 6 which lists two possible resource combinations for a task with streaming input and non-streaming output. While combination 6(a) has slower resources (note that the reservation interval lengths are longer), it is still preferable to 6(b) because it has an earlier completion time.

TieBestCombinationInserter (TBCI) This approach improves BestCombinationInserter by handling tie situations. If two combinations have the same completion time, then the selection is made based on the total cost of each combination, and vice versa.

3.3 Handling constraint violations

Only one strategy to handle constraint violation situations is presented here, due to space limitations, although others are available. The DeleteViolatingWorkflow (DVW) strategy simply blacklists the offending task's workflow and removes all reservations that have already been made for that workflow. This enforces strict constraints, and violating workflows are removed to free up resources.

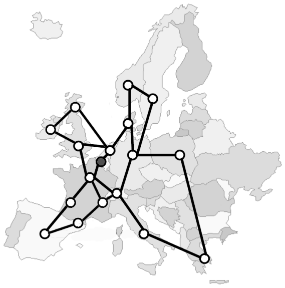


Fig. 7. The European network as a result of the COST-292 project

4 Results and Discussion

4.1 Developed simulation environment

Testlab evaluations and field trial implementations require a lot of effort, and are impractical to perform large-scale testing. Only rigorous testing on large-scale environments with different setups can provide a true comparison on the performance of different algorithms. Since building large testbeds is costly, time consuming and often impractical, simulation is an important aid in the evaluation of algorithms.

Therefore, a simulation framework is designed and implemented in Java to facilitate the evaluation of the developed algorithms. It is able to simulate any combination of algorithm, network topology and workflow load. The configuration of network infrastructure, workflow load and the scheduling algorithm to use are supplied to the simulator in an XML format. Note that the workflow load supplied to the simulator is a fixed description consisting of a list of workflows including individual submit times and properties such as deadline and budget. This allows us to evaluate different scheduling strategies on the same resource topology and workflow load.

For performance reasons, the simulator performs high-level simulation of network transfers (i.e. not down to packet level but transfer times determined based on data size, network route and individual network link bandwidth). The simulator logs detail information on all tasks and resources.

4.2 Simulation Setup

The following evaluation setup has been used to obtain the results presented here. A simulation is run for each algorithm using the same load and network. These simulations are repeated a number of times with different loads and networks to obtain a view on the average performance of the algorithms.

The basic network topology is randomly generated starting from the European network shown in Figure 7 but with varying parameters such as available network bandwidth between nodes, the number, speed and cost of Computational Resources and the number and data repositories of the Data Resources.

Bandwidth between cities is randomly chosen from 1Gbps, 100Mbps or 10Mbps. Cities have a number of nodes connected to them - always through 1Gbps links - on which the Computational and Data Resources reside. Data and Computational Resources never share a node, and each city has at least one Data Resource and one Computational Resource. Three classes of Computational resources exist, executing tasks at 0.5x, 1.0x or 2x reference speed, and for the simulations presented here, tasks can execute on every Computational Resource.

All workflows are line workflows, i.e. there are no parallel or alternative paths. Tasks' input and output data sizes range from 768Mb to 46Gb, according to a random uniform distribution. Input and output can be streaming or non-streaming. Processing time depends on the size of the input data and the Computational Resource's speed. Processing 768Mb data at reference speed takes 120 seconds, while processing 46Gb takes 2 hours at reference speed. A task has a probability of 0.3 to find its requested input data on a particular Data Resource. Likewise, it has a 0.3 probability to be able to store its output data on a particular Data Resource.

Three different scenarios are evaluated, with workflow loads consisting of respectively 250, 500 and 750 workflows over a 24h interval, with a uniform distribution arrival. These loads are chosen to stress the available resources and the algorithms.

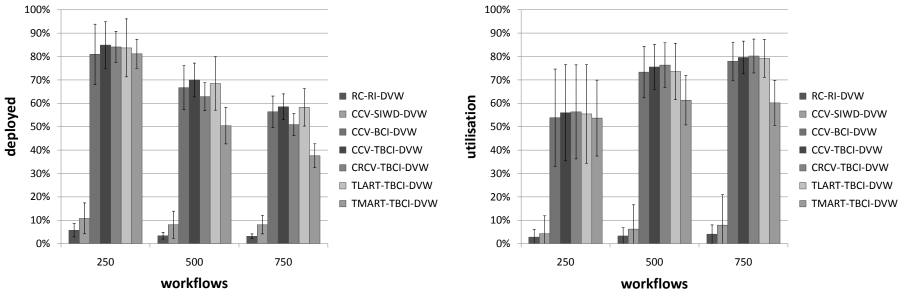
4.3 Detailed evaluation results

The results on the following combinations of priority-, assignment-, and constraint handling strategies are presented: RC-RI-DVW, CCV-SIWD-DVW, CCV-BCI-DVW, CCV-TBCI-DVW, CRCV-TBCI-DVW, TLART-TBCI-DVW and TMART-TBCI-DVW. The abbreviations for the different strategies are defined in Section 3.

Figure 8(a) shows the average workflow acceptance rates for 20 different runs, as a function of an increasing workflow load. While these loads may not seem very high, recall that the execution of a single task can take several hours. Half of the runs prioritized makespan, the other half budget. The workflow acceptance rate is the percentage of workflows that can be successfully assigned to resources and executed. Recall that strict constraints are enforced, i.e. workflows that would violate a constraint are simply not started.

Figure 8(b) - which shows the average Computational Resource occupation as a function of an increasing load - serves to illustrate that the decrease in acceptance rates as loads go up is attributed to a saturation of Computational Resources, rather than the algorithms. Note that it is very unlikely for resources to reach full utilisation, due to free time fragmentation on the one hand, and the dependencies between reservations on different resources on the other hand.

From Figure 8(a), it is evident that randomly assigning workflows to resources results in unacceptable workflow acceptance rates. Likewise, using the naive SIWD resource assignment strategy produces barely higher acceptance rates. SIWD is inefficient because it considers Computational and Data Resources independently, which has several implications. Firstly, choosing a fast



(a) Deployment rate

(b) Computational Resource occupation

Fig. 8. Average results over 20 runs including standard deviation, as a function of an increasing workflow load. Loads are over a 24h interval. The standard deviations are shown as well.

Computational Resource independently makes no guarantees about the available Data Resources and network links for that resource. Furthermore, always choosing the best Computational Resource ensures that these Computational Resources will quickly become unavailable.

Further observations show that the tie considering strategy of TBCI indeed improves the BCI strategy slightly, as it produces a higher acceptance rate.

The results show that the performance of the CRCV priority heuristic compared to the CCV approach drops as the loads go up. The rationale behind the CRCV priority strategy is to give higher priority to workflows that have already used up relatively more of their budget or available time. However, the absolute time or budget remaining is important, as it allows CCV to assign lower priorities to tasks and workflows with loose constraints and high priority to workflows with strict constraints. This is however not the case for CRCV. This can result in situations where workflows or tasks with loose constraints are given priority and thus needlessly reserve fast or cheap resources, while they could have completed with slower or more expensive resources.

The TMART priority algorithm performs considerably worse than TLART. Giving more priority to tasks that are unlikely to be deployed definitely increases the possibility of these tasks being deployed, without adversely affecting tasks that already have a good possibility of getting deployed. The opposite 'greedy' strategy TMART definitely has a negative impact on the amount of workflows that can be deployed.

Figure 9 displays the running times of the algorithms to determine a schedule, as a function of an increasing workflow load. The algorithms are executed in Sun Java 6 on an AMD Opteron 2350 machine with 8Gb memory installed. The less complex SIWD and BCA assignment strategies are obviously considerably faster than the BCI and TBCI strategies, which need to consider every resource combination for each task. Furthermore, the more complex priority assignment

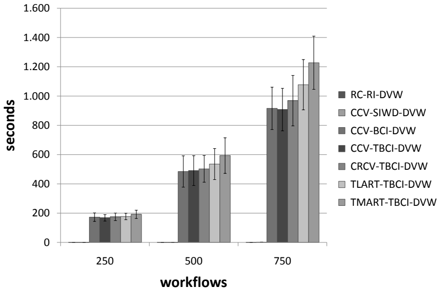


Fig. 9. Average execution time of each algorithm as a function of the load, 24h interval. The standard deviations are shown as well.

algorithms TLART and TMART also cause higher execution times, as they need to evaluate the occupation of every potential resource for each task.

The higher execution time of TMART versus TLART can be attributed to the fact that TMART can deploy considerable less workflows than TLART. Consequently, the constraint violation handler is called more often for TMART, resulting in an increased execution time.

5 Future Work

Areas for future work include a more extensive evaluation on the performance of the various algorithms, not limited to line workflows, but also including completely arbitrary workflows containing parallel paths and alternative paths of decision branches.

Additionally, only one constraint handling strategy has been presented in this paper. Additional constraint violation handling strategies are available, and their evaluation must be presented as well. Alternate (meta-)heuristics may also be evaluated.

Furthermore, online scheduling algorithms have been designed, and their detailed performance evaluation is ongoing at the time of writing.

6 Conclusions

The Independent Films In Progress (IFIP) project is a framework aimed at the media industry, providing a platform for users to automate and submit production processes such as editing and distribution. Users submit a series of tasks or workflows they want to be executed within certain constraints. The platform autonomously assigns the appropriate resources to the users' workflows and executes them. A key issue in such a platform is how to efficiently allocate tasks to the available resources, hereby improving throughput, scalability and QoS.

To this end, this paper presents a number of offline scheduling algorithms which are based on the List Scheduling technique, but use different strategies to handle the various stages of the heuristic such as assigning priorities to tasks, task-to-resource assignments and constraint violation handling. Extensive evaluation of the developed algorithms has proven that they outperform naive implementations, and shows the importance of selecting adequate priority, deployment and constraint handling strategies.

Acknowledgment

Part of the research described in this paper is funded through the IBBT-project IFIP.

References

1. IBBT: Independent films in progress (IFIP). `http://www.ibbt.be/en/project/ifip` (2008-2010)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). W. H. Freeman (January 1979)
3. Yu, J., Buyya, R., Ramamohanarao, K.: Workflow scheduling algorithms for grid computing. In: Metaheuristics for Scheduling in Distributed Computing Environments. Springer (2008)
4. Dong, F., Akl, S.G.: Scheduling algorithms for grid computing: State of the art and open problems. Technical report, School of Computing, Queen's Univeristy (2006)
5. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4) (1999) 406–471
6. Andries, L.: Facing media traffic challenges. Broadcast Engineering (Feb, 2010)
7. IBBT: Grid enabled infrastructure for service oriented high definition media applications (GEISHA). `http://www.ibbt.be/en/project/geisha`
8. Volckaert, B., Wauters, T., De Leenheer, M., Thysebaert, P., De Turck, F., Dhoedt, B., Demeester, P.: Gridification of collaborative audiovisual organizations through the mediagrid framework. Future Gener. Comput. Syst. **24**(5) (2008) 371–389
9. Harmer, T.: Gridcast–a next generation broadcast infrastructure? Cluster Computing **10**(3) (2007) 277–285
10. Walsh, A.E.: The media grid: A public utility for digital media. In: 8th International Symposium on Spatial Media (ISSM) jointly held with IEEE International Conference on Computer and Information Technology (IEEE CIT). (2007)
11. Basu, S., Adhikari, S., Kumar, R., Yan, Y., Hochmuth, R., Blaho, B.E.: mmgrid: Distributed resource management infrastructure for multimedia applications. In: IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, Washington, DC, USA, IEEE Computer Society (2003) 88.1
12. Seinstra, F.J., Geusebroek, J.M., Koelma, D., Snoek, C.G.M., Worring, M., Smeulders, A.W.M.: High-performance distributed video content analysis with parallelhorus. IEEE MultiMedia **14**(4) (2007) 64–75
13. Chrétienne, P., Coffman, E.G., Lenstra, J.K., Liu, Z., eds.: Scheduling Theory and its Applications. John Wiley and Sons (1995)
14. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. **33**(6) (June 2007) 369–384
15. Alhusaini, A.H., Prasanna, V.K., Raghavendra, C.S.: A unified resource scheduling framework for heterogeneous computing environments. In: Proc. Eighth Heterogeneous Computing Workshop (HCW '99). (April 12, 1999) 156–165