Scalable Grid Application Scheduling via Decoupled Resource Selection and Scheduling

Yang Zhang*, Anirban Mandal*, Henri Casanova‡,
Andrew A. Chien†, Yang-Suk Kee†, Ken Kennedy*, Charles Koelbel*
*Computer Science Department
Rice University
{yzhang8,anirban,ken,chk}@cs.rice.edu
†Dept. of Computer Science and Engineering
University of California at San Diego
{achien,yski}@ucsd.edu
‡Information and Computer Sciences Department
University of Hawai'i at Manoa
{henric}@hawaii.edu

***Abstract*— Over the past years grid infrastructures have been deployed at larger and larger scales, with envisioned deployments incorporating tens of thousands of resources. Therefore, application scheduling algorithms can become unscalable (albeit polynomial) and thus unusable in large-scale environments. One reason for unscalability is that these algorithms perform implicit resource selection. One can achieve better scalability by performing explicit resource selection independently from scheduling in a "decoupled" approach. Furthermore, we hypothesize that one can achieve similar or even better performance as with the non-decoupled approach, which we call the "one step" approach, by selecting resources judiciously. Leveraging the Virtual Grid abstraction, we demonstrate that the decoupled approach is indeed both scalable and effective in large-scale and highly heterogeneous resource environments.**

I. Introduction

Grids [11], [12] are attractive platforms for deploying parallel and distributed applications at large scale and/or high performance. One key issue for achieving these goals is that of application scheduling, that is the decision process by which application components are allocated to available resources to optimize some metric of performance. Scheduling parallel and distributed applications is a well-known difficult problem [13] and many scheduling heuristics have been proposed in the literature (see [17] and [5] for surveys of scheduling heuristics for applications with and without task dependencies). Scheduling applications onto grid platforms poses new challenges and has been studied for many relevant application classes [6], [21], [9]. One distinguishing feature of grid platforms is the large number of individual resources, with current systems containing thousands of resources and future systems incorporating tens or hundreds of thousands. This volume of resources raises scalability issues (e.g., resource discovery, resource monitoring). In this paper we specifically address the scalability of the scheduling algorithm itself: how can one compute an application schedule in a short amount of time while considering a large number of potential resources?

We observe that although the resource environment may contain large numbers of resources, all taken into consideration when computing a schedule, typically only a small subset of these resources is used for running the application. In essence, most scheduling heuristics typically perform *implicit resource selection*: the set of resources used by the application emerges from the computation of the schedule. In this paper we improve the scalability of the scheduling process by performing *explicit resource selection*: in contrast to the traditional *one-step* approach, which considers all available resources when scheduling, we use a *decoupled* approach, which selects the resources for consideration first and then schedules the application on these resources.

We use the *Virtual Grid* (VG) abstraction, which we described in previous work [16]. A VG provides a high-level, hierarchical abstraction of the resource collection that is needed and used by an application. A user creates a VG specification, written in the Virtual Grid Description Language (vgDL), and passes it to the Virtual Grid Execution System (vgES). The vgES performs fast resource selection in grid environments with hundreds of thousands of resources, returning a set of selected physical resources on which one can schedule the application. The set of selected resources is typically many orders of magnitude smaller than the whole universe of resources, and the running time of a scheduling algorithm over this smaller subset of resources is also orders of magnitude shorter.

While decoupling resource selection from scheduling in large-scale systems as described above clearly improves scalability, a key question is: what is the impact of decoupled resource selection and scheduling on the quality of the resulting schedule? In this paper we study decoupled resource selection and scheduling in the context of

workflow applications in large-scale highly heterogeneous grid environments and make three contributions:

1) We demonstrate how the VG abstraction can be leveraged to decouple resource selection and application scheduling in a generic way (i.e., our approach is in principle applicable to any scheduling algorithm and any grid application).
2) One key issue in our decoupled approach is that of choosing an appropriate resource selection methodology. We discuss and provide a quantitative evaluation of several factors that affect the construction of an appropriate VG specification.
3) Using simulations of representative workflow applications on representative grid environments, we quantify the trade-off between scalability and schedule quality for our decoupled approach, demonstrating that it achieves schedule quality comparable to that achieved by one step approaches, at dramatically higher scalability.

This paper is organized as follows. In Section II we present our decoupled resource selection and scheduling approach for large-scale Grid environment. Section III discusses our resource selection strategy, by introducing the Virtual Grid concept, the specific scheduling and selection methodologies used and the application context in which we evaluate it. Experimental evaluation and results are presented in Section IV. Section V discusses related work, and Section VI concludes the paper with a summary of contributions and perspectives on future work.

II. Decoupled Scheduling Approach

The process of scheduling a parallel application on a distributed platform can be described as follows. Given an application that consists of m "tasks" (e.g., computations, I/O operations), and given a platform that consists of n "resources" (e.g., CPUs, disks, networks), compute a mapping of tasks to time and to resources (i.e., task i starts executing at time t on resource j). This mapping is called a schedule, and the "scheduling problem" consists in computing a schedule that optimizes some metric, such as overall application execution time (or makespan). The scheduling problem has been studied in many contexts (with task dependencies [17], with independent tasks [5], etc.), is known to be NP-complete in most of its instances [13], and many polynomial-time scheduling heuristics have been proposed.

When there are many heterogeneous resources and/or when the application has fewer concurrent tasks than there are resources, many resources are unused. In our experience, this is usually the case when scheduling scientific applications onto large-scale grid platforms. In fact, most applications use only a very small fraction of all available resources. In such cases, we say that the heuristic performs *implicit resource selection*. Results presented in this paper show that, on platforms that incorporate this many resources, the time to run even a linear-time heuristic can be prohibitive, making application scheduling unscalable.

A solution to address this scalability problem consists in decoupling resource selection from application scheduling. In the first phase, one performs *explicit resource selection*. In the second phase, one performs scheduling within the selected resources rather than on the whole resource universe. The key point here is that a decoupled approach makes it possible to compute schedules faster, by several orders of magnitude, making application scheduling scalable to large-scale platforms. In fact, this decoupling may make it possible to run expensive scheduling algorithms on the explicitly selected resources.

We claim that using a system such as vgES (see Section III-A) to perform explicit resource selection makes it possible to achieve schedules that are comparable in quality to the ones obtained when letting the scheduling algorithms perform implicit resource selection over the whole resource universe, at dramatically higher scalability. Although our decoupled approach is generic, in this paper we discuss and evaluate it in the context of a particular class of applications, as seen in Section III-D.

III. Decoupled Application Scheduling in Grid Environments

A. Virtual Grid and Resource Selection

A fundamental challenge for grid applications is to describe and obtain appropriate resources to enable efficient, high performance execution. This is challenging from many standpoints, including the definition of an appropriate abstraction to describe resource needs, the difficulty of finding appropriate resources quickly in an environment with many thousands of resources, and interacting with diverse, autonomous resource managers that implement their own resource management and access policies. As noted in the introduction, the VGrADS project [16] approaches this by allowing the user to specify its resource needs using a high-level language, vgDL [8], which our execution system, vgES [16], uses to find and allocate appropriate resources for the application, returning a VG abstraction, which is really an active entity (i.e., runtime object). By contrast with traditional low-level resource description and selection systems [3], [2] that focus on individual, quantitative resource characteristics, the VG provides a high-level, hierarchical abstraction of the resource collection that is needed by an application. The application can then use the VG to find specific information about the allocated physical resources, to deploy application components, and to modify or evolve the resource collection.

We refer the reader to [16], [8] for details regarding the vgES system and we only describe here features of vgDL that are relevant for this work. The vgDL language uses high-level resource abstractions that correspond to what grid application programmers typically use to organize their applications portably across many different resource environments. The vgDL was designed based on

a detailed study of half a dozen real-world applications. These studies showed that in order to design for performance (and to manage complexity) portably, application developers typically use three simple resource abstractions to aggregate individual resources. Consequently, vgDL contains three resource aggregates, distinguished based on homogeneity and network connectivity: (i) LooseBag — a collection of heterogeneous resources with no good connectivity guaranteed; (ii) TightBag — a collection of heterogeneous resources with good connectivity; (iii) Cluster — adding homogeneity, a well-connected set of resources with identical (or nearly so) attributes. Each aggregate specifies a range for its size (i.e., number of resources). The user can specify constraints on attributes of individual resources within the aggregate (e.g., clock rate, processor architecture, memory, etc.), or constraints on aggregate attributes (e.g., total aggregate memory, total aggregate disk space). Aggregates can be nested (e.g., a LooseBag of Clusters) to arbitrary depth. With these resource aggregate abstractions, an application can structure the specification of its resource environment in a top-down fashion and decorate components with constraints when desired. In addition to constraints, applications can also express resource preference by using a scalar rank function: a user-defined expression of basic arithmetic operators, resource attribute and resource aggregate attribute values that define a scalar value that represents the quality of that resource set for the application's request.

The Virtual Grid Execution System (vgES) uses efficient search techniques based on resource classification in a relational database. Table indices and other more sophisticated database optimization techniques make the search highly scalable in environments with large number of resources. For instance, Figure 1 shows that it takes no more than 5 seconds for the vgES system to process one million resources for various queries on a Pentium4 3.2 Ghz processor. The different lines in the figure represent different types of query with L, T, C meaning LooseBag, TightBag and Cluster respectively and the number denoting the size of the requested VG. We will see in Section IV that the ability to perform such resource selection in a few seconds is key for improving the scalability of application scheduling on large-scale platforms.

Given that (i) vgDL makes it possible to specify high-level, qualitative resource requirements and that (ii) vgES can perform fast resource selection in large-scale resource environments, the VGrADS project provides an ideal foundation for decoupling resource selection from application scheduling.

B. Scheduling Algorithms

While our decoupling approach is applicable to any scheduling algorithm, we chose to apply it to a specific workflow-scheduling algorithms to evaluate our approach.

In this paper, we use a simple greedy workflow scheduling scheme, which works as follows. Until all tasks in the workflow are scheduled, the algorithm finds the current set of ready tasks. Then is calculates the rank values for each of these available tasks on each available resource. The rank values reflect the expected performance of the tasks on the resources, with lower being better. Rank values take into account the expected cost of computation and data movement. Once the rank values are obtained, for each available task to schedule, the algorithm assigns it to the resource for which the task has the minimum estimated completion time (ECT), which is a function of the rank and the current state and availability of the resources. The computational complexity of our greedy scheduling scheme is $O(vp)$ in which v denotes the number of jobs and p denotes the number of resources.

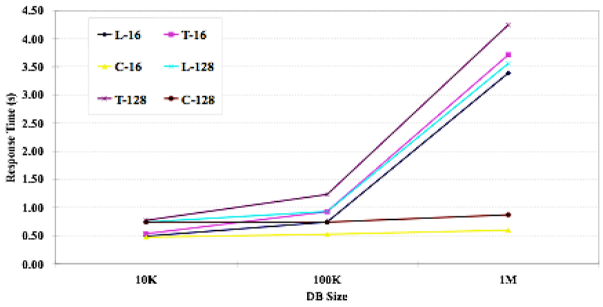

Fig. 1. Time to complete vgDL queries with vgES.

We have also experimented with various heuristics [5] which have computational complexity of $O(v^2p)$. We do not report those results here because the algorithms are less scalable, and because for the DAGs in Section III-D they produce nearly identical schedules to the greedy method. However, in future work we plan to conduct experiments with more complex applications for which the benefit of more complex schedulers may be greater.

C. Selection Methodology

Now that we have picked a scheduling algorithm, we must decide on a resource selection strategy. Resource selection must be done according to the application's needs and we consider three classes for three different types of such needs:

1) **Class 1**: A set of resources that have high computing power but not necessarily good network connection between them, as needed by a computation intensive applications.
2) **Class 2**: A set of resources that are connected with high bandwidth and low latency but are not necessarily have high computing power, as needed by a communication intensive application.
3) **Class 3**: A set of resources that have relatively balanced computing power and connectivity, as needed by a balanced application that is neither compute- nor communication-intensive.

It is relatively straightforward to generate selection criteria for class 1: simply select the fastest resources. However, we need the help of vgDL specifications and of the vgES system to select the resources that meet the requirement of class 2 and 3. The key concept here is the TightBag. Recall from Section III-A. that a TightBag is a collection of heterogeneous nodes with good connectivity. It matches the requirement of class 2 perfectly. For class 3 we will use vgDL to specify a hierarchy of aggregates. The idea is to aggregate several TightBags into a single LooseBag so that we can get both high computation power and high connectivity.

The above classes provide bases for performing resource selection following three broad characterizations of an application's resource needs. In our experiments we quantify the needs of particular applications using a popular and simple metric: the application's Communication to Computation Ratio (CCR). We experiment with applications spanning a spectrum of CCR values, with higher values implying that the application is communication-intensive and lower values implying that the application is computation-intensive. For each such application we perform resource selection according to the three above classes. We expect that class 1 will be best for applications with low CCRs, and that class 2 will be best for applications with high CCRs. We will verify that the CCR value of the application provides good guidance for selecting the appropriate resource selection method.

The final key element for resource selection is the specification of a bound on the number of required resources. One could ask for as many (potential) resources as there are resources in the whole universe of resources. This will not lead to any scalability improvement over a traditional application scheduling approach that performs implicit resource selection. Instead, as a simple heuristic, we request as many resources as the maximum width of the DAG representing the application's workflow. The intuition behind this choice is that this is the maximum number of resources that can be used by the application at a given time. Any additional resource may stay idle for the entire application execution.

D. Case-Study: Workflow Applications

We explore our approach of decoupled resource selection and scheduling in the context of two real applications, EMAN [19] and Montage [24]. These applications fall into the general class of workflow applications. We choose workflow applications for our representative scenarios because they are amongst the most popular grid applications today.

A workflow application consists of a collection of interacting components that need to be executed in a certain partial order for successful execution of the application. The components have specific control and data dependence between them. Hence, the application can be represented as a Directed Acyclic Graph (DAG), where each node in the DAG represents an application component and the edges denote control/data dependencies. We describe our two target applications below.

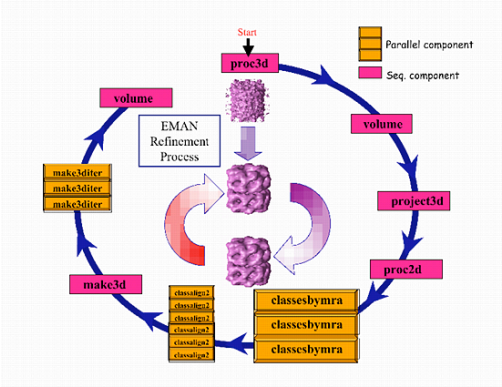

Fig. 2. EMAN Refinement Workflow

1) EMAN: EMAN [Electron Micrograph Analysis] is a bio-imaging application developed at the Baylor College of Medicine. It primarily deals with 3D reconstruction of single particles from electron micrographs. Human expertise is needed to construct a preliminary 3D model from the 'noisy' electron micrographs. The refinement from a preliminary 3D model to the final 3D model is fully automated and is the most computationally intensive step that benefits from harnessing the power of the grid. The EMAN refinement can be represented by the workflow depicted in Figure 2. It is essentially a linear workflow with some sequential and parallel stages. The important and time-consuming steps are the large parameter sweep steps like "classesbymra". We use different versions of the EMAN refinement workflow DAG for the experiments.

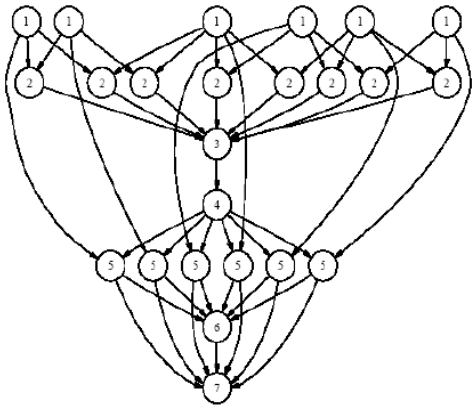

Fig. 3. A Small Montage Workflow

2) Montage: Montage is a data-intensive astronomy application to create custom image mosaics of the sky on demand. It consists of four steps: (i) Re-projection of input images to a common spatial scale; (ii) Modeling of

background radiation in images to achieve common flux scales and background levels; (iii) Rectification of images to a common flux scale and background level; and (iv) Co-addition of re-projected, background-corrected images into a final mosaic. Figure 3 shows the structure of a small Montage workflow. The workflow consists of some highly parallel sections that can benefit from execution over multiple grid sites. This application being data-intensive, potentially large files are transferred on the edges of the workflow. We use different versions of this workflow for our experiments.

IV. Experimental Evaluation

A. Methodology

1) Simulation Environment: In order to perform repeatable experiments in a large-scale resource environment we resort to simulation. Our simulated environment consists of three key components: the resource model, the network model, and the application model.

Our resource model is based on a tool that generates populations of representative compute clusters [15]. This tool uses empirical statistical models of cluster characteristics (e.g., number of processors, processor architecture, processor clock rate) obtained from a survey of 114 real-world clusters. Using this tool we generated a resource pool that contains over 36,000 hosts, which we call the *resource universe*.

Our network model is as follows. Conforming to the results in [18], we generated end-to-end latencies between compute clusters according to a truncated normal distribution. We set the mean of this distribution to 100ms, conforming to the results in [22], and we bounded the latencies from 1 to 200ms. For the network bandwidths, we set the connection within a cluster as 1000Mb/s and all the inter connection between clusters range from 10Mb to 100Mb/s. These numbers are primarily based on results listed in [27], [10]. Furthermore, we ensured that the higher the latency the lower the bandwidth.

Our application model comes directly from the real-world applications described in Section III-D. For each application we generate DAGs that follow the same structure as those of the applications, but we vary their CCR and their widths. When simulating application execution, the execution times of the tasks on resources come from the DAG node weights and the performance models described in [20], and the data transfer times come from the DAG edge weights and the latencies and bandwidth in our network model.

Since this is a simulated environment, we must make some assumptions that may not hold for real resources. We assume that we have an accurate performance model for tasks for both scheduling and computing the simulated makespan. (In fact, we have such models for EMAN and Montage.) We argue that since both the one-step and the decoupled scheduler use the same performance model, this does not bias the comparison. We assume that the network performance is stable and predictable. This assumption eliminates the random error that may be introduced by the network fluctuation. We have found it to be the case for our experiments with EMAN, although longer-running applications would see more variation. We assume that the resource is available immediately, and will remain available for the duration of the application. We assume that we already obtained all the resource information before the start of the experiment. Once again, we believe that these assumptions do not bias our comparison between the two scheduling approaches.

We plan future experiments to relax these assumptions for more in-depth study of scheduling methods. In particular, performance variation of the tasks or networks may require new scheduling algorithms, but those can be applied to either the one-step or decoupled approach. Dynamic resource availability requires dynamic rescheduling (as we studied in the GrADS project [**?**]) or reselection of resources. In either case, vgDL and vgES will support such updates for the decoupled approach.

2) Experimental Setup: We first generate forty EMAN and Montage DAGs with five different CCRs and eight different widths. We use the greedy scheduling algorithm described in Section III-B to schedule these DAGs on the simulated resources. For each DAG, we first run the scheduling algorithm on the whole resource universe, which we refer to as the *one-step approach*, and record the running time of the scheduler. We then run the scheduling algorithm on smaller subsets of resources explicitly selected using the methodologies in Section III-C. The running time for this *decoupled approach* is measured as the sum of the time for selection and time to compute the schedule. In both cases we record the (simulated) makespan of the application. To run our experiments, we used the Rice Terascale Cluster which is composed of Intel 900 MHz Itanium2 machines [26].

In order to determine how resource selection affects scheduler performance, we selected 10%, 7%, 3%, 1% and 0.3% of the "best" resources, corresponding to the resource selection methods for class 1 in Section III-C. We will later refer these as the *simple selection approach*. We also performed selections based on vgDL specifications. To satisfy the requirements of class 2, we generated the vgDL description shown in Figure 4, requesting one TightBag of OPTERON and ITANIUM nodes. The "[1:n]" means there are at most "n" nodes in the TightBag; we substituted the maximum DAG width for n. We will later refer to this selection methodology as the *one TightBag approach*. Similarly, we generated the vgDL description shown in Figure 5 for class 3. We will later refer to this selection methodology as the *LooseBag approach*. Finally we refer to the implicit resource selection approach used by the one-step approach as the *Universe approach*.

```
VG = TightBagOf(node)[1:n] {
    node = [(Processor == OPTERON) || (Processor== ITANIUM )]
           [Rank=Clock]
}
```

Fig. 4. vgDL for class 2 type of resource abstraction

```
VG = LooseBagOf(tb)[1:n] {
    tb = TightBagOf(node)[1:500] [Rank= Nodes] {
      node = [
               (Processor == OPTERON) || (Processor== ITANIUM )
             ]
             [
                 Rank=Clock
             ]
    }
}
```

Fig. 5. vgDL for class 3 type of resource abstraction

B. Results

Figures 6 and 7 show the average scheduler running time of the one-step and decoupled approaches for EMAN and Montage. From the graphs we can see that, even including selection time, the scheduler's total running time is linear in the number of resources considered. Even in this case, the time used in the decoupled approaches is only a small fraction of the time used in the one-step approach, since the number of selected resources is much lower than the full grid. This confirms our hypothesis of better scalability of the decoupled approach.

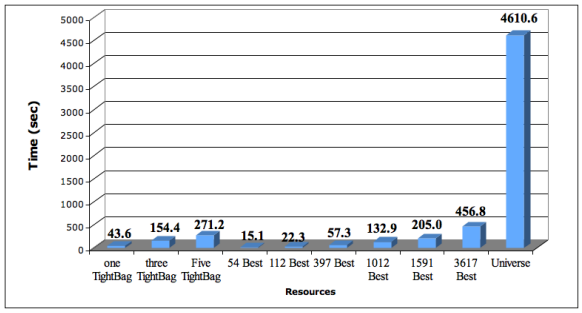

Fig. 6. Average Scheduling+Selection Time for EMAN DAGs

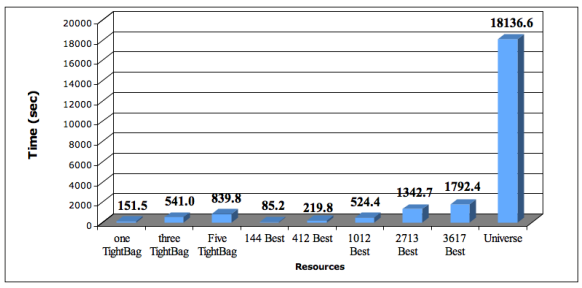

Fig. 7. Average Scheduling+Selection Time for Montage DAGs

Figures 8, 9, and 10 show the combined makespan (yellow) and scheduling (blue) time for a range of simulations. In all charts, the total *turnaround* time for the application is the overall height of the bar. For the "Simple Selection", "One Tightbag", and "LooseBag" bars, we used the scheduling time for the case in Figures 6 and 7 that selects the least resources more than the maximum width of the DAG. For example, for a test DAG of width 518, the Simple bar uses the "1012 Best" scheduling time, the TightBag bar uses "One TightBag", and the LooseBag bar uses "Three TightBags". All results are averages over a collection of EMAN and Montage DAGs.

Figure 8 shows results for computation-intensive DAGs belonging to class 1. We observe that all the decoupled approaches have much better turnaround time compared to the one-step approach. Among decoupled approaches, the one TightBag approach performs the worst since it does not provide enough computing power. The simple selection approach performs the best with makespan only 2% worse than the one-step approach. This confirms our hypothesis that simple selection is very suitable for these applications.

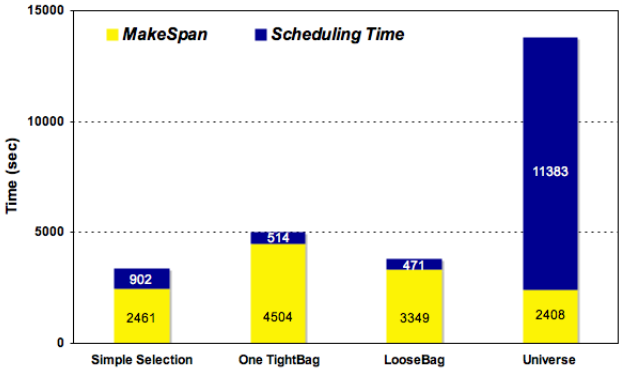

Fig. 8. Average MakeSpan and Scheduling Time for DAGs with CCR=0.1

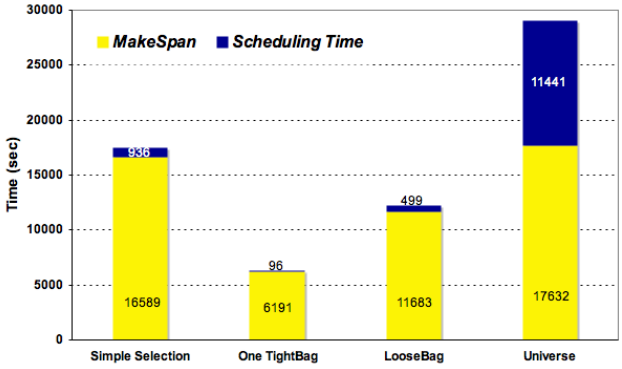

Fig. 9. Average MakeSpan and Scheduling Time for DAGs with CCR=10

Figure 9 shows results for communication-intensive

DAGs belonging to class 2. We observe that all decoupled approaches have lower turnaround time than the one-step approach. The one TightBag approach has the best performance and outperforms the one-step approach by almost 66%. The main reason for this result is that all selected resources are closely connected, which avoids greedily choosing nodes with poor connectivity. A better scheduling heuristic for the Universe case might reduce its Makespan, but at the cost of even higher scheduling time. This confirms our hypothesis that pre-selecting a TightBag is appropriate and efficient for scheduling this class of problems.

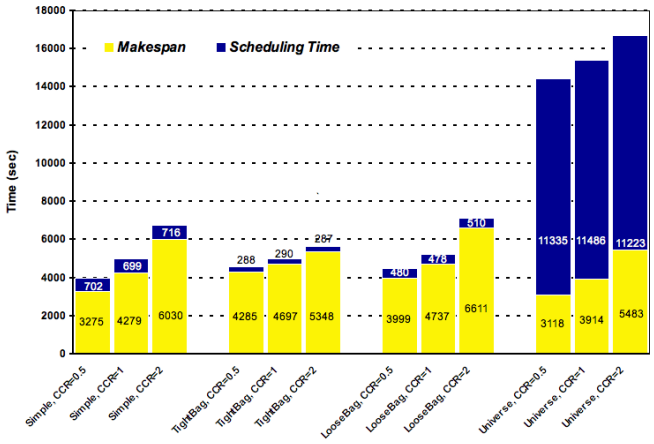

Fig. 10. Average MakeSpan and Scheduling Time for DAGs with CCR=0.5,1,2

Figure 10 shows results for DAGs with relatively balanced communication and computation requirements, such as those in class 3. In all cases, the decoupled approaches have lower turnaround times than the one-step approach due to their lower scheduling times, with gains of up to 50%. Also as we expected, the Simple approach performs better as the CCR gets below one (i.e. more computation-intensive code) and the TightBag approach performs better as the CCR gets above one. Unexpectedly however, the LooseBag approach does not show a clear advantage. Here are two possible reasons

1) The simple selection may implicitly select nodes that are close since fast nodes are more likely found in a few clusters than scattered around the grid.
2) The bandwidth between the TightBags within the LooseBag we choose may happen to be very low.

We will investigate this further in the future. If reason 1 is true, we can further simplify our VGDL requests, while if reason 2 is the case, we may have to devise more complex queries.

In summary, our experiments confirm our hypothesized advantages for decoupled scheduling over the one-step approach. They also confirm our hypotheses of best scheduling methods for very computation- and communication-intensive applications. However, they do not match our expectations for balanced applications.

V. Related Work

This work is closely related to our previous work on scheduling Directed Acyclic Graphs onto heterogeneous platforms under VGrADS [16] project. Current grid workflow management systems use simple approaches such as first-come-first-served with matchmaking as in Condor DAGMan [21], the Data Grid resource broker [28] and the GridLab resource broker [1], or random allocations or round robin as in Pegasus [25].

Mandal et al.[20] presents a novel approach to schedule the nodes in a DAG level by level. The algorithm relies on a performance model to assign a rank to each node-resource pair and then uses a combination of three heuristics to schedule the nodes in the same level onto those resources. Blythe et al.[4] improves the algorithm by introducing randomness as the authors have observed that a better local (i.e., within a level) makespan may not always lead to a better DAG makespan. The algorithm repeatedly computes a new mapping between nodes and resources at each level to find a better overall makespan. Evaluation results show that this approach works well for communication-intensive DAGs but not as well for computation-intensive ones. A key limitation of this approach however is that it is not scalable to large numbers of resources as task-resource mappings are re-computed repeatedly.

VI. Conclusion

In this work, we have presented a decoupled mechanism that leverages the concept of a Virtual Grid to schedule workflow applications onto large-scale grid environments. Our approach improves scalability when compared to traditional scheduling approaches as schedules can be computed dramatically faster. Furthermore, our experimental results show that our approach does not increase the makespan significantly in practice. Therefore, our approach dramatically improves application turnaround time.

In future work we will conduct experiments for a wider range of DAGs including randomly generated ones. We will also apply and evaluate our decoupled approach to more scheduling algorithms, such as HEFT [14] and DLS [23]. Finally, we plan to conduct experiments on real resources including our own VGrADS testbed and the TeraGrid [7].

At a more fundamental level, this work can be extended in several directions. For instance, at the moment the granularity of our resource abstraction category is relatively coarse and we assume that we can always find enough resources to match our resource selection criteria. This may not be the case in practice and we need to refine our resource selection criteria when we cannot find enough resources (e.g., ask for fewer resources, ask for slower resources, ask for resources with poorer connectivity). Also, the CCR is calculated using average network

bandwidth, which may not work when networks are highly heterogeneous.

Acknowledgments

This work was supported by the National Science Foundation grant 0331645. Access to the RTC was supported by NSF grant 0216467, Intel, and HP.

References

[1] G. Allen, D. Angulo, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzysky, J. Pukacki, M. Russell, T. Radke, E. Seidel, J. Shalf, and I. Taylor. GridLab: Enabling Applications on the Grid. In *GRID '02: Proceedings of the Third International Workshop on Grid Computing*, pages 39–45, London, UK, 2002. Springer-Verlag.

[2] G. Avellino, S. Barale, S. Beco, B. Cantalupo, D. Colling, F. Giacomini, A. Gianelle, A. Guarise, A. Krenek, D. Kouril, and A. Maraschini et al. The EU DataGrid Workload Management System: towards the second major release. 2003.

[3] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crumme, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *Int. J. High Perform. Comput. Appl.*, 15(4):327–344, 2001.

[4] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task Scheduling Strategies for Workflow-based Applications in Grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE Press, 2005.

[5] T. Braun, H. Siegel, and N. Beck. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.

[6] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. GridFlow: Workflow Management for Grid Computing. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 198, Washington, DC, USA, 2003. IEEE Computer Society.

[7] Charlie Catlett and et al. http://www.teragrid.org/about, 2002.

[8] A. Chien, H. Casanova, Y.-S. Kee, and R. Huang. The Virtual Grid Description Language: vgDL. . Technical Report CS2005-0817, University of California, San Diego, Department of Computer Computer Science and Engineering, Aug 2005.

[9] H. Dail, H. Casanova, and F. Berman. A decoupled scheduling approach for the GrADS program development environment. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–14, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[10] A. Denis, O. Aumage, R. Hofman, K. Verstoep, T. Kielmann, and H. E. Bal. Wide-Area Communication for Grids: An Integrated Solution to Connectivity, Performance and Security Problems. In *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, pages 97–106, Washington, DC, USA, 2004. IEEE Computer Society.

[11] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kauffmann Publishers, Inc., 1999.

[12] I. Foster and C. Kesselman. *The Grid2.* Morgan Kauffmann Publishers, Inc., 2003.

[13] M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

[14] H.Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 2(13):260–274, 2002.

[15] Y.-S. Kee, H. Casanova, and A. A. Chien. Realistic Modeling and Svnthesis of Resources for Computational Grids. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 54, Washington, DC, USA, 2004. IEEE Computer Society.

[16] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, and A. Chien. Efficient Resource Description and High Quality Selection for Virtual Grids. In *Proceedings of the 5th IEEE Symposium on Cluster Computing and the Grid (CCGrid'05), Cardiff, U.K.*, May 2005.

[17] Y. Kwok and I. Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms . *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.

[18] C. Lee and J. Stepanek. On future Global Grid communication Performance. *Heterogeneous Computing Workshop*, 2001.

[19] S. Ludtke, P. Baldwin, and W. Chiu. EMAN: Semiautomated software for high resolution single-particle reconstructions. *J. Struct. Biol*, (128):82–97, 1999.

[20] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling Strategies for Mapping Application Workflows onto the Grid. In *14-th IEEE Symposium on High Performance Distributed Computing (HPDC14)*, pages 125–134, 2005.

[21] Dagman MetaScheduler. http://www.cs.wisc.edu/condor/dagman.

[22] Robert Morris and et al. http://pdos.csail.mit.edu/p2psim/kingdata, 2004.

[23] G. C. Sih and E. A. Lee. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4(2):175–187, 1993.

[24] G. Singh, E. Deelman, and G. Bruce Berriman et al. Montage: a Grid Enabled Image Mosaic Service for the National Virtual Observatory. *Astronomical Data Analysis Software and Systems*, (13), 2003.

[25] G. Singh, E. Deelman, G. Mehta, K. Vahi, M.-H. Su, G. B. Berriman, J. Good, J. C. Jacob, D. S. Katz, A. Lazzarini, K. Blackburn, and S. Koranda. The Pegasus portal: web based grid computing. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 680–686, New York, NY, USA, 2005. ACM Press.

[26] Rice University. http://cohesion.rice.edu/centersandinst/citi/research.cfm?doc_id=5949.

[27] L. Yang, J. M. Schopf, and I.Foster. Improving Parallel Data Transfer Times Using Predicted Variances in Shared Networks. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE Press, 2005.

[28] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, T. Jordan, E. Quigg, and J. Dobson. Grid middleware services for virtual data discovery, composition, and integration. In *Proceedings of the 2nd workshop on Middleware for grid computing*, pages 57–62, New York, NY, USA, 2004. ACM Press.