What makes workflows work in an opportunistic environment?

Ewa Deelman[1] Tevfik Kosar[2] Carl Kesselman[1] Miron Livny[2]

[1]USC Information Science Institute, Marina Del Rey, CA

deelman@isi.edu, **carl@isi.edu**

[2]Computer Sciences Department, University of Wisconsin, Madison, WI

kosart@cs.wisc.edu, miron@cs.wisc.edu

Abstract

In this paper, we examine the issues of workflow mapping and execution in opportunistic environments such as the grid. As applications become ever more complex, the process of choosing the appropriate resources and successfully executing the application components becomes ever more difficult. In this paper, we focus on the interplay between a workflow mapping component that plans the high-level resource assignments and the workflow executor that oversees the component execution. We concentrate particularly on issues of data management and we draw from the experiences with mapping and execution systems: Pegasus, DAGMan and Stork.

1 Introduction

Many scientific applications today are being developed as complex workflows, where the workflow steps represent individual application components and the dependencies in the workflow impose precedence on the application component execution. Workflows enable scientists to systematically express complex analysis, reason about the overall application and to provide provenance information adequate for the interpretation of the derived results. As the complexity of the applications grows, so does the need to use a significant number of resources to support their execution. It is often the case, that no single group of collaborators has all the computational resources in their possession and may form larger collaborations to draw upon a larger set of common resources. These resources are no longer dedicated to one application or one user group; rather they can be opportunistically used by multiple groups when available. Recently many domain scientists in high-energy physics [CMS][ATLAS], gravitational-wave physics [LIGO] and astronomy [SDSS][DPOSS] have been turning towards opportunistic environments

such as the grid to enable day-to-day large-scale data analysis. However, such environments pose a significant challenge: resources are shared among many users, policies governing their use may change over time, hardware and software failures may occur. When applications are complex, being able to bring an application to a successful completion is difficult. It is nearly impossible for a user to map and then manage the execution of the application by hand. Users often rely on various middleware services to perform many of the functions. Some can map a workflow-based application onto available resources and some can make sure that the resulting instantiated workflow components are executed in a prescribed order. The complexity arises when faults occur in the system, and the middleware components need anticipate and/or recover from them. Dealing with failures may often involve more than one middleware services. In this paper, we focus on a particular problem of the interplay between decision-making and execution services: the planner and the manager. We draw from our experiences with the Pegasus planner [Deelman 03, Deelman 03b, Deelman 04] and a manager that consists of DAGMan [DAGMan], Condor-G [Frey 01] and Stork [Kosar 04] will be used to discuss the limitation of current tools and describe possible ways to enhance the interaction between these software tools.

2 Background

Much of our experience stems from our work within the NSF-funded Grid Physics Network (GriPhyN) project [Griphyn]. GriPhyN focuses on supporting a variety of large-scale applications such as CMS [CMS] and Atlas [Atlas] (high-energy physics), SDSS [SDSS] (astronomy) and LIGO [LIGO] (gravitational-wave physics). At the heart of GriPhyN is the idea of virtual data where data can exist in a materialized form (accessible from some storage system) or can exist in a form of a recipe (or workflow). When a user request a data set, the system (composed of several services) evaluates the request and generates an abstract workflow, performs the necessary resource assignment and executes the request.

As part of GriPhyN, several middleware services have been developed that can take a high-level partial workflow description, map it to a concrete form and execute on the grid. In particular, the Chimera system takes a Virtual Data Language (VDL) provided by the user and constructs an abstract workflow. The abstract workflow details the application components and their input and output data at an abstract level—without specifying the resources that will be used in the execution or the specific location of the

data. Pegasus takes that abstract description, queries a variety of grid information services and makes decisions about where to execute the application components and where to access the data. Pegasus may also decide to reduce the abstract workflow if intermediate data products are already available. In order to stage data in and out of the application components, Pegasus augments the workflow with data movement. The resulting concrete workflow is then given to a workflow execution system (or workload manager) such as DAGMan. The workload manager interacts with a variety of resource managers that control the allocation of these physical resources. For simplicity, we also assume that the workflows are structured as directed acyclic graphs (DAGs). A directed graph manager takes the workflows, orders the jobs according to their dependencies, and submits the jobs ready for execution to the corresponding batch schedulers. Different batch schedulers can be used according to the requirements or characteristics of the jobs in the workflow. In our framework, we use two specialized batch schedulers: one for computation (DAGMan) and one for data placement (Stork).

3 Workflow Mapping and Execution

Figure 1 gives an overview of the workflow mapping and execution components. The main interaction between the planner and the manager is a concrete DAG. In this DAG, jobs are represented as nodes and the dependencies between jobs are represented as directed arcs between the respective nodes. To perform the management of the DAGs, we employ the Directed Acyclic Graph Manager (DAGMan) which is a service for executing multiple jobs with dependencies between them. DAGMan accepts a declaration that specifies the jobs to be executed and the order of their execution. It logs the execution of the DAG to persistent storage, allowing it to resume a DAG where it left off, even in the face of crashes and other failures.

The Condor [Litzkow 88] workload scheduling system is a scheduler for computational jobs. Condor provides a job queuing mechanism and resource monitoring capabilities. It allows the users to specify scheduling policies and enforce priorities. Condor has an extension called Condor-G, which allows users to submit their jobs to inter-domain resources by using the Globus Toolkit [Foster 99] functionality. In this way, user jobs can be scheduled and run not only on Condor resources but also on PBS [Henderson 96], LSF [Zhou 92], LoadLeveler [IBM 96], and other grid resources.

We have used Stork as the scheduler for data transfer jobs. Stork is a specialized

scheduler for data placement activities in heterogeneous environments. Data placement comprises all data movement related activities such as transfer, staging, replication, space allocation and de-allocation. Stork can queue, schedule, monitor, and manage data placement jobs and ensure that the jobs complete.

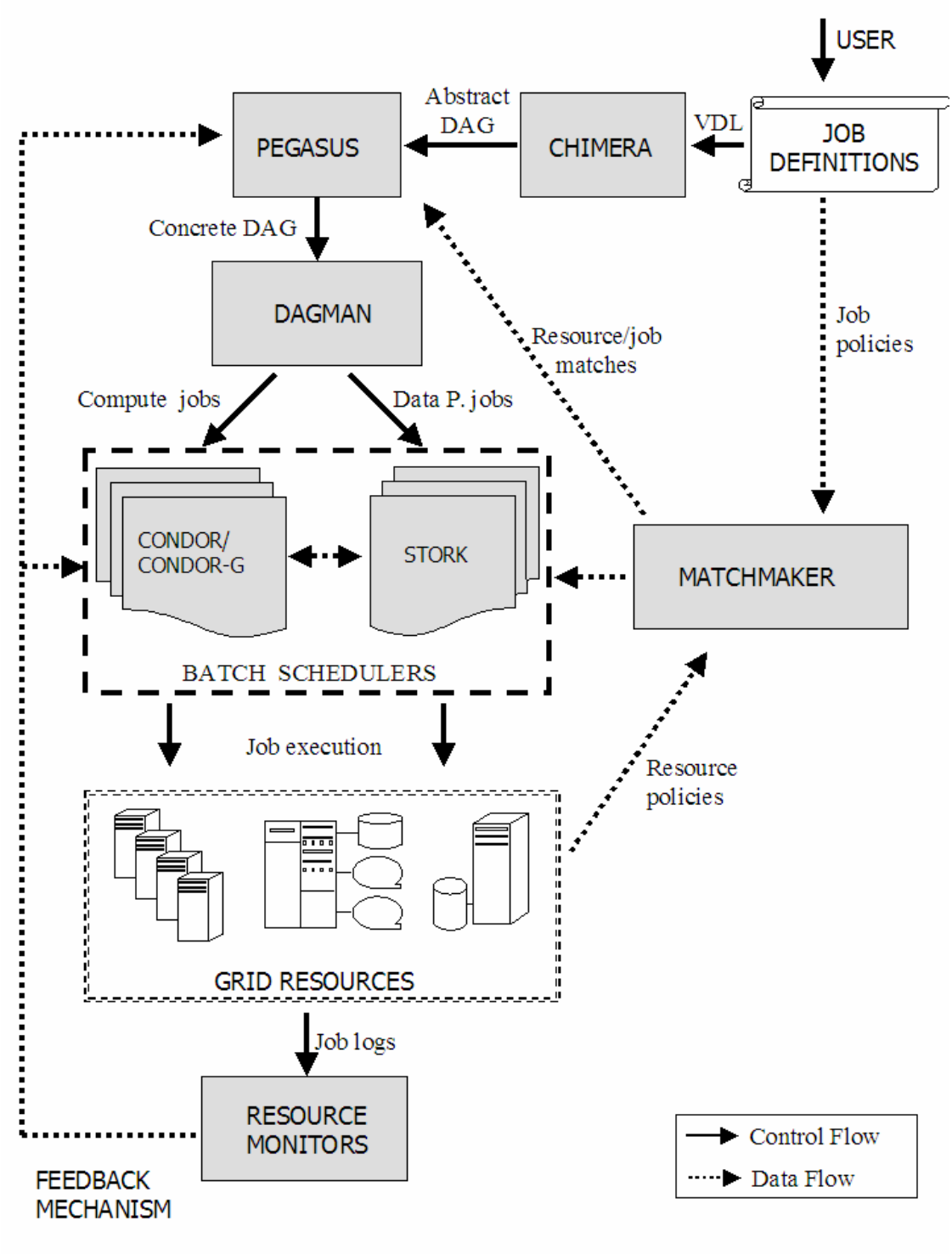

Figure 1: Overall view of workflow management components.

4 Data Management issues

Data-intensive applications in distributed computing systems may require moving the input data for the job from a remote site to the execution site, executing the job, and then moving the output data from execution site to the same or another remote site. These data movement decisions are made in our system by Pegasus. In order to avoid the risk of running out of disk space at the execution site, Pegasus may also allocate space before transferring the input data there, and release the space after it moves out the output data from there. We regard all of these computational and data placement steps as real jobs and represent them as nodes in a DAG as shown in Figure 2.

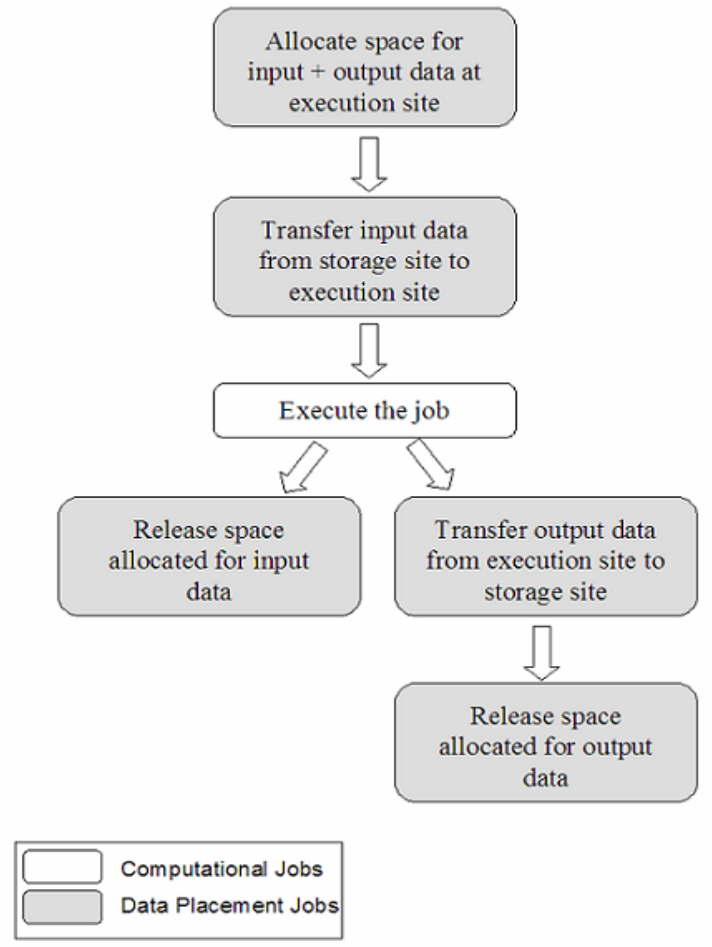

Figure 2: Separation of data placement from computation.

Data placement jobs are represented in a different way than computational jobs in the job specification language, so that the high-level planners can differentiate these two classes of jobs. Then, the planner submits computational jobs to a compute job queue, and the

data placement jobs to a data placement job queue. Jobs in each queue get scheduled by the corresponding scheduler.

Since our focus in this work is on the data placement part, we do not get into details of the computational job scheduling. The data placement scheduler can understand the characteristics of the data placement jobs and can make smart scheduling decisions accordingly. Computational job schedulers do not understand the semantics of data transfers well.

For example, if the transfer of a large file fails, we do not want simply restart the job and re-transfer the whole file. Rather, we may prefer to transfer only the remaining part of the file. Similarly, if a transfer using one protocol fails, we may want to try other protocols supported by the source and destination hosts to perform the transfer. We also may want the scheduler to choose and apply the network tuning parameters such as I/O block size, TCP buffer size and number of parallel streams, which best fits to the selected data transfer protocol. A traditional computational job scheduler does not handle these cases. For this purpose, we differentiate the data placement jobs from computational jobs.

The data placement component schedules the jobs in its queue according to the information it gets from the high-level planner and from the resource broker/policy enforcer (matchmaker). The resource broker matches resources to jobs, and helps in locating the data and making decisions such as where to move the data. The policy enforcer helps in applying the resource specific or job specific policies, such as how many concurrent connections are allowed to a specific storage server.

The transfer history data collected from the job log files and the network statistics from the network monitoring tools are fed back to the scheduler and the high-level planner. The job description can be changed according to this feedback as well whenever it is necessary.

5 Limitations of the current approach

When failures in the environment are infrequent, the workflow can be simply passed from the planning component to the execution component. However, in the face of potentially many failures, the simple delegation scenario may not be sufficient. Failure handling for computational jobs is fairly straightforward if they do not have any data dependencies. The most common failure handling mechanism for these types of jobs is simply rescheduling to another computational resource and retrying. If the jobs are dependent on the existence or the transfer of some data, in other words data placement, more complicated failure recovery mechanisms need to be employed depending on the failure reason.

If the reason of failure for the computational job is some missing input data, the earlier jobs in the workflow which were responsible of creating and transferring the necessary input data for this particular jobs may need to be re-executed as well. This can involve either the planner or the workflow manager. The workflow manager can simply do a rollback in the workflow, and then repeat some of the already successfully completed jobs to meet the failed job's requirements. Sometimes, even a rollback may not be sufficient. The planner may need to be involved and it may need to change the workflow or reconsider site and replica selection if necessary.

If the job fails due to insufficient disk space, either it should be assigned to another resource with sufficient disk space, or space reservations should be made to ensure that adequate disk space remains during a file transfer or during creation of output data by a computational job.

All of these actions require coordination between the planner, workflow manager and the corresponding batch schedulers. Such interactions are currently not available in today's GriPhyN systems.

6 Interactions between Workflow Planning and Execution

In this section, we abstract away from the details of the systems we presented so far and focus on the interactions that are necessary between workflow planning and execution components. As we already mentioned, in order for the abstract workflow to be executed on opportunistic distributed resources, such as for example resources provided by a Grid, a mapping from the abstract workflow specification to actual resources has to be performed throughout the execution of the workflow. In general, the mapping can be viewed as an iterative process. The planner devises a plan and delegates it to the manager for execution. Given the evolution of the execution of the plan, it may be necessary to adjust the mapping decisions. The level of specification of the plan may vary depending on the capabilities of the planner and the manager. In general, because of the highly dynamic aspects of the underlying execution environment, it may be beneficial to postpone the binding of a task to a physical resource until the very last moment.

6.1 Planning decision points

Not all decisions can be postponed to the end. The manager needs to have a general idea of the work that needs to be managed. The planner needs to construct a high-level plan for the entire workflow ahead of time and provide the manager with a general structure of the workflow. The planner may augment the initial abstract workflow with additional activities that may be necessary for the actual execution, such as resource allocation and de-allocation, stage-in and stage-out, and clean up after the job is finished, as shown in figure 3. The planner may also reduce the workflow by removing activities whose result is already available. For example, if several jobs requiring the same input file get assigned to the same execution site, some of the steps for storage allocation and data transfers can be merged as shown in figure 4. The initial workflow adaptation may also depend upon the current state of the execution environment. The final decision of allocation of a particular activity may occur at three distinct points in time.

Workflow Delegation Time: Decisions can be made at the time that the execution of the workflow is delegated to the manager (*eager planning*). In eager planning, the planner makes the decision given the state of the resources at the time just prior to the delegation.

Activity Scheduling Time: Although the manager is provided with the structure of the

workflow at delegation time, the decision of resource assignment is made when a particular activity or activity set is ready to be released into the system (in particular when all the parents of the activities are successfully completed). In this mode, that we term *deferred planning*, the information used during the mapping reflects the state of the execution system at the time that an activity can be released to the execution environment.

Resource Availability Time: It is possible that at the time an activity is ready to execute, there are no resources available to execute that activity or that it is difficult to determine the "best" resources at that time. *Just-in-time* planning allows decisions to be made when resources needed to execute the activity become available.

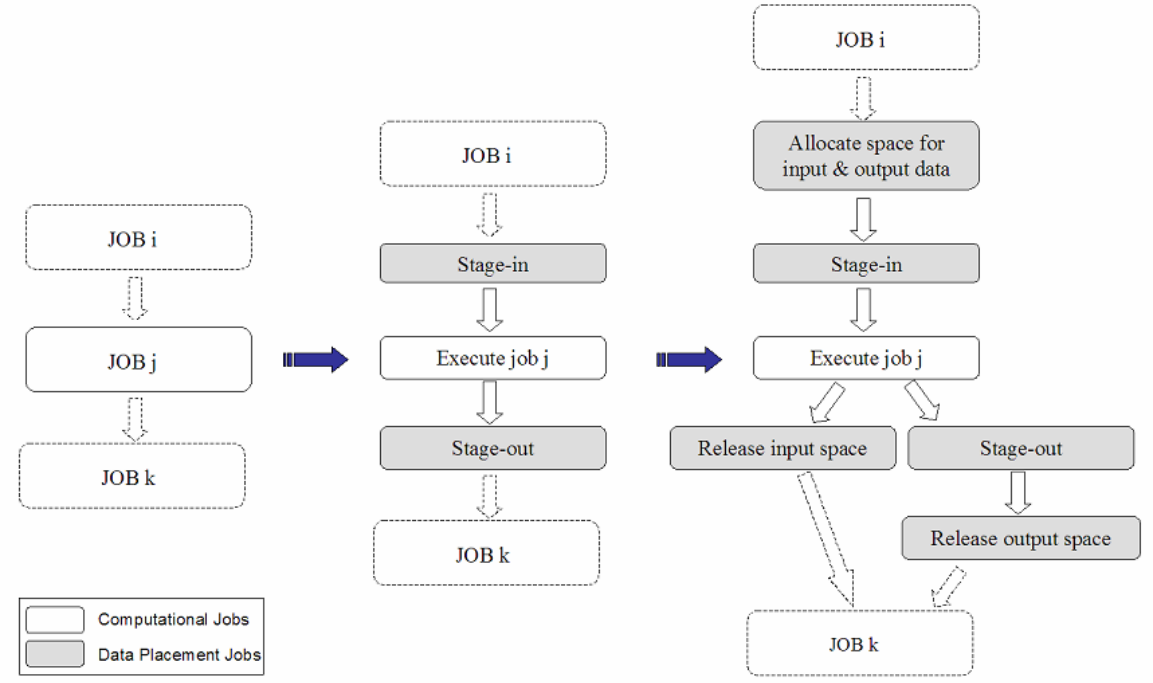

Figure 3: Extending the DAG according to the requirements of the job.

Although, in general it may be advantageous to schedule as late as possible, it may sometimes be also detrimental. For example, if a computation requires a large input data set, it can be beneficial to pre-stage the data to a particular resource and then execute there. That approach is better than picking an available computation resource as it becomes available and then stage the data in.

Also, the entire workflow may not need to be treated uniformly. It is possible that some

planning can and needs to be done ahead of time, for example if the workflow can use resources that support reservation, or when we are dealing with a reasonably stable execution environment, for example a Grid that has been dedicated for a particular time, where resource contention is low. In the case that reservations are not available, but we are dealing with rare resources, such instruments or environments composed of queuing-based resources (for example the TeraGrid) it may be advantageous to use the deferred approach. Just-in-time planning may be beneficial in a highly dynamic environment where resources can suddenly become available or go away and where resource contention is high.

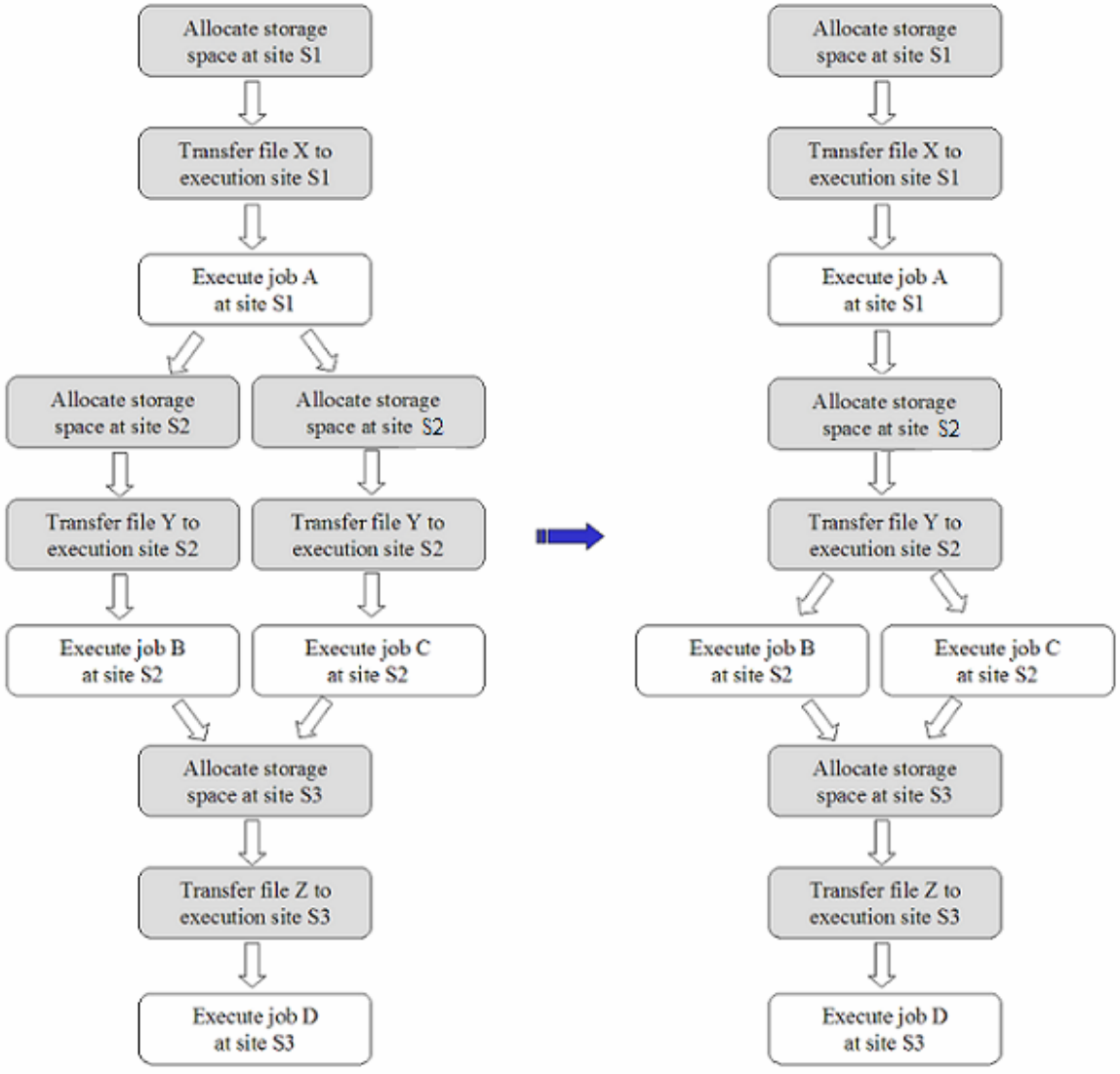

Figure 4: Reduction of unnecessary nodes in the DAG.

As one can imagine, a particular workflow can benefit from a variety of scheduling

paradigms, reserving resources ahead of time where possible, submitting jobs into queues of high-performance resources, hoping that resources become available and taking advantage of them. The interactions between the planner and manager need to support that type of flexible resource assignment. Additional complexity arises from the fact that resources may suddenly disappear. In that case previously made decisions may need to be revisited. We touch upon some of these issues in the Section 6.3.

6.2 Decision Specification Level

The level of specification that the planner gives to the manager is dependent on the capabilities of each of these components. On one end of the spectrum, the planner may be able to structure the workflow but then push all the decision making to the manager. This requires a manager that is capable of making decisions (good or bad). On the other end of the spectrum, we can have a manager with no decision-making capabilities where we need to rely on the planner to make all the decisions. In practice, the capabilities of the planners and managers may be somewhere in between, enabling a certain level of sophistication in the decision making process by both components.

Notice that the capabilities of the planner and manager do not necessarily dictate any particular decision points rather they dictate the level of specification that the planner gives to the manager. For example, we can imagine a very sophisticated planner. Let us assume that we plan eagerly. If the planner has at its disposal a simple manager, it will need to tell that manager exactly what resources to use at each step of the workflow. However, if the manager is more sophisticated, rather than receiving directives from the planner, it can receive “guidance” in the form of preferred resources, policies to use when making resource assignments, etc. The final, deferred, or just-in-time decision can be left to the manager.

Having a simple manager does not imply that the decision cannot be made in the deferred mode. The directives coming from the planner can instruct the manager to call back to the planner at the time when that activity is ready to be released into the execution system. The just-in-time mode is a bit more complex, because it relies on the capability of the manager to recognize the opportunities presented by the environment (new resources suddenly becoming available).

6.3 Reacting to the Changing Environment and Recovering from Failures

Interactions between the planner and the manager are not solely geared towards delegating work from the planner to the manager. In a reliable and static environment, the initial plan (at any level of specificity) can be given for execution by the manager. However, distributed environments, especially the Grid, are very dynamic and prone to failure. One cannot rely solely on the manager or the execution environment to handle failures and to adapt to changing resource loads. Although the manager may try to recover from failures by methods such as retry, it may ultimately fail to execute the specified task. It is thus necessary for the workflow manager to communicate the failure, as well as success to the planner. We term this process as "flow back" denoting the flow back of the information from the manager to the planner. Based on the information provided by the manager, the planner can decide how to proceed, whether to reschedule a particular activity and possibly its dependents.

Again, the level of sophistication of the planner and manager play an important role. It determines how much recovery is placed within the responsibility of the components. If the manager is sophisticated, it can possibly try to schedule to a different set of resources (based on its own knowledge or based on the planner directives). If the manager is not that sophisticated or has tried all that it could, it may want to communicate the failures back to the planner. However, if the planner is not very sophisticated, it may not attempt recovery procedures beyond those used by the manager. In any case, the planner is ultimately responsible for the end-to-end execution of the workflow and needs to communicate with higher-level systems (maybe a user).

The planner may be able to come up with a different set of directives, possibly even re-computing parts of the workflow that were not exposed to the manager. For example, the planner may have reduced the abstract workflow based on the assumption that particular data products were available in the environment. If for example, these products are suddenly not available, either through resource failure or through user interaction (deleting a file); the planner may augment the workflow and include the generation of the necessary data. In general, in case of failures, the planner may want to re-plan the mapping of the workflow.

7 Related work

There have been a number of efforts within the Grid community to develop general-purpose workflow management solutions.

WebFlow [Fox 98] is a multileveled system for high performance distributed computing. It consists of three layers. The top layer consists of a web based tool for visual programming and monitoring. It provides the user the ability to compose new applications with existing components using a drag and drop capability. The middle layer consists of distributed web flow server implemented using java extensions to httpd servers. The lower layer uses the Java CoG Kit to interface with the Grid [Laszewski 01] for high performance computing. Webflow uses GRAM as the interface between Webflow and the Globus Toolkit. Thus, Webflow also provides a visual programming aid for the Globus toolkit.

GridFlow [Cao 03] has a two-tiered architecture with global Grid workflow management and local Grid sub workflow scheduling. GridAnt [gridant] uses the Ant [ant] workflow processing engine. Nimrod-G [Buyya 00] is a cost and deadline based resource management and scheduling system. The Accelerated Strategic Computing Initiative Grid [Beiriger 00] distributed resource manager includes a desktop submission tool, a workflow manager and a resource broker. In the ASCI Grid software components are registered so that the user can ask "run code X" and the system finds out an appropriate resource to run the code. Pegasus uses a similar concept of virtual data where the user can ask "get Y" where Y is a data product and the system figures out how to compute Y. Almost all the systems mentioned above except GridFlow use the Globus Toolkit [Foster 99] for resource discovery and job submission. The GridFlow project will apply the OGSA [ogsa] standards and protocols when their system becomes more mature. Both ASCI Grid and Nimrod-G uses the Globus MDS [Fitzgerald 97] service for resource discovery and a similar interface is being developed for Pegasus. GridAnt, Nimrod-G and Pegasus use GRAM [Czajkowski 98] for remote job submission and GSI [Welch 03] for authentication purposes. GridAnt has predefined tasks for authentication, file transfer and job execution, while reusing the XML-based workflow specification implicitly included in ant, which also makes it possible to describe parallel and sequential executions.

The main difference between Pegasus and the above systems is that while most of the above system focus on resource brokerage and scheduling strategies Pegasus uses the concept of virtual data and provenance to generate and reduce the workflow based on data products, which have already been computed earlier. It prunes the workflow based

on the assumption that it is always more costly the compute the data product than to fetch it from an existing location. Pegasus also automates the job of replica selection so that the user does not have to specify the location of the input data files. Pegasus can also map and schedule only portions of the workflow at a time, using just in-time planning techniques.

BAD-FS [Bent 04] builds a batch aware distributed filesystem for data intensive workloads. This is general purpose and serves workloads more data intensive than conventional ones. For performance reasons it prefer to access source data from local disk rather than over a network filesystem. Further, BAD-FS at present does not schedule wide-area data movement which we feel is necessary for large data sets. BAD-FS can interact with our system for more reliable and efficient wide area transfers.

Pasquale et al [Pasquale 94] present a framework for operating systems level I/O pipelines for efficiently transferring very large volumes of data between multiple processes and I/O devices. This work is at very low level and does not apply to distributed systems.

Application Level Schedulers (AppLeS) [Berman 96] have been developed to achieve efficient scheduling by taking into account both application-specific and dynamic system information. AppLeS agents use dynamic system information provided by the NWS [Wolski 97].

Beck et. al. introduce Logistical Networking [Beck 00] which performs global scheduling and optimization of data movement, storage and computation based on a model that takes into account all the network's underlying physical resources. This approach only focuses only on the data movement and does not perform any coordination of CPU and data.

Thain et. al. propose the Ethernet approach [Thain 03] to distributed computing, in which they introduce a simple scripting language which can handle failures in a manner similar to exceptions in some languages. The Ethernet approach is not aware of the semantics of the jobs it is running; its duty is retrying any given job for a number of times in a fault tolerant manner

8 Conclusions

In summary, the process of mapping workflows in distributed, opportunistic environments requires a dynamic multi-stage approach. There is a need to have a mapping component that can make decisions on the global scale—the scale of the entire workflow and a component that can react and adapt to the changing environment. These

components cannot be independent of one another, rather they need to be closely coordinated. In this paper we use our recent experience in managing real-life workflows on Grid resources [Griphyn] to identify challenges and discuss possible approaches to address them. The ideas presented in this paper will be used by the authors and their teams to enhance the functionality of their respective software tools.

References:

[ATLAS] CERN, “A Toroidal LHC ApparatuS Project”, http://atlas.web.cern.ch/Atlas.

[Beck 00] M. Beck, T. Moore, J. Plank and M. Swany, “Logistical Networking”, *Active Middleware Services*, S. Hariri, C. Lee and C. Raghavendra, editors, Kluwer Academic Publishers, 2000.

[Bent 04] J. Bent, D. Thain, A. Arpaci-Dusseau and R. Arpaci-Dusseau, “Explicit Control in a Batch-Aware Distributed File System”, *In the Proceedings of the First USENIX/ACM Conference on Networked Systems Design and Implementation (NSDI 2004)*, March, 2004.

[Beiriger 00] Beiriger, J., Johnson, W., Bivens, H., Humphreys, S. and Rhea, R., Constructing the ASCI Grid. In Proc. 9th IEEE Symposium on High Performance Distributed Computing, 2000, pp. 193-200. IEEE Press.

[Berman 96] F. Berman, R. Wolski, S. Figueira, J. Schopf and G. Shao, “Application Level Scheduling on Distributed Heterogeneous Networks”, *In Proceedings of Supercomputing'96*, Pittsburgh, PA, November 1996.

[Buyya 00] Buyya, R., Abramson, D. and Giddy, J. "Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid", HPC Asia 2000, May 14-17, 2000, pp 283 - 289, Beijing, China.

[Cao 03] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. GridFlow: WorkFlow Management for Grid Computing. In Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03), pp. 198-205, 2003.

[Czajkowski 98] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. A Resource Management Architecture for Metacomputing Systems. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.

[CMS] CERN, “The Compact Muon Solenoid Project”, http://cmsinfo.cern.ch.

[DAGMan] Condor Team. “The directed acyclic graph manager (DAGMan)”, http://www.cs.wisc.edu/condor/dagman, 2002

[Deelman 03] E. Deelman, J. Blythe, et al., "Mapping Abstract Complex Workflows onto

Grid Environments," Journal of Grid Computing, vol 1, nr 1, pages 25-29, 2003.

[Deelman 03b] E. Deelman, J. Blythe, Y. Gil, Carl Kesselman "Workflow Management in GriPhyN", Chapter in "The Grid Resource Management", 2003.

[Deelman 04] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, "Pegasus: Mapping Scientific Workflows onto the Grid", *To appear in the Proceedings of across Grid EU Conference*, 2004.

[DPOSS] S. G. Djorgovski, R. R. Gal, S. C. Odewahn, R. R. de Carvalho, R. Brunner, G. Longo and R. Scaramella, "The Palomar Digital Sky Survey (DPOSS)", *Wide Field Surveys in Cosmology*, 1988.

[Fitzgerald 97] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. Proc. 6th IEEE Symposium on High-Performance Distributed Computing, pp. 365-375, 1997.

[Foster 99] I. Foster and C. Kesselmann, "Globus: A Toolkit-Based Grid Architecture", *The Grid: Blueprints for a new Computing Infrastructure*, pages 259-278, Morgan Kaufmann, 1999.

[Fox 98] http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Akarsu809

[Frey 01] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: AComputation Management Agent for Multi-Institutional Grids.," *Cluster Computing*, vol. 5, pp. 237-246, 2002.

[gridant] http://www-unix.globus.org/cog/projects/gridant

[Griphyn] Griphyn, "Grid Physics Network", http://www.griphyn.org

[Henderson 96] R. Henderson and D. Tweten, "Portable Batch System: External Reference Specification", 1996.

[IBM 96] IBM, "Using and Administering IBM Load Leveler", *IBM Corporation SC23-3989*, 1996.

[Kosar 04] T. Kosar and Miron Livny, "Stork: Making Data Placement a First Class Citizen in the Grid", *In the Proceedings of 24th IEEE International Conference on Distributed Computing Systems (ICDCS 2004)*, Tokyo, Japan, March 2004.

[Laszewski 01] von Laszewski, G. I. Foster, J. Gawor, P. Lane, "A Java Commodity Grid Kit**,"** *Concurrency and Computation: Practice and Experience*, pages 643-662, Volume 13, Issue 8-9, 2001.

[LIGO] Caltech, "Laser Interferometer Gravitational Wave Observatory", http://www.ligo.caltech.edu.

[Litzkow 88] M. J. Litzkow, M. Livny and M. W. Mutka, "Condor – A Hunter of Idle Workstations", *In Proceedings of the 8th Int. Conf. of Distributed Computing Systems,*

pages 104-111, 1988.

[ogsa] www.globus.org/ogsa

[Pasquale 94] J. Pasquale and E. Anderson, “Container Shipping: Operating System support for {I/O} Intensive Applications”, *IEEE Computer*, 1994.

[SDSS] FNAL, “Sloan Digital Sky Survey”, http://tdpc01.fnal.gov/sdss/project.htm.

[Thain 03] D. Thain and M. Livny, “The Ethernet Approach to Grid Computing”, *In Proceedings of the Twelfth IEEE Symposium on High Performance Distributed Computing (HPDC-12)*, Seattle, WA, June 2003.

[Welch 03] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Cajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke. Security for Grid Services. Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), pp. 48 -57, June 2003, IEEE Press.

[Wolski 97] R. Wolski, “Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service”, *In Proceedings of the Sixth IEEE Symposium on High Performance Distributed Computing (HPDC6)*, Portland, OR, August 1997.

[Zhou 92] S. Zhou, “LSF: Load Sharing in Large-Scale Heterogeneous Distributed Systems”, *In Proceedings of Workshop on Cluster Computing*, 1992.