

Chapter 1

TEN ACTIONS WHEN GRID SCHEDULING

The User as a Grid Scheduler

Jennifer M. Schopf

Mathematics and Computer Science Division, Argonne National Laboratory

Abstract In this chapter we present a general architecture or plan for scheduling on a Grid. A Grid scheduler (or broker) must make resource selection decisions in an environment where it has no control over the local resources, the resources are distributed, and information about the systems is often limited or dated. These interactions are also closely tied to the functionality of the Grid Information Services. This Grid scheduling approach has three phases: resource discovery, system selection, and job execution. We detail the steps involved in each phase.

1. INTRODUCTION

More applications are turning to Grid computing to meet their computational and data storage needs. Single sites are simply no longer efficient for meeting the resource needs of high-end applications, and using distributed resources can give the application many benefits. Effective Grid computing is possible, however, only if the resources are scheduled well.

Grid scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites. We define a *job* to be anything that needs a resource – from a bandwidth request, to an application, to a set of applications (for example, a parameter sweep). We use the term *resource* to mean anything that can be scheduled: a machine, disk space, a QoS network, and so forth. In general, for ease of use, in this chapter we refer to resources in terms associated with compute resources; however, nothing about the approach is limited in this way.

In general we can differentiate between a Grid scheduler and a *local resource scheduler*, that is, a scheduler that is responsible for scheduling and

managing resources at a single site, or perhaps only for a single cluster or resource. These are the lowest-level of lower-level scheduling instances discussed in Chapter ?? One of the primary differences between a Grid scheduler and a local resource scheduler is that the Grid scheduler does not “own” the resources at a site (unlike the local resource scheduler) and therefore does not have control over them. The Grid scheduler must make best-effort decisions and then submit the job to the resources selected, generally as the user. Furthermore, often the Grid scheduler does not have control over the full set of jobs submitted to it, or even know about the jobs being sent to the resources it is considering use of, so decisions that tradeoff one job’s access for another’s may not be able to be made in the global sense. This lack of ownership and control is the source of many of the problems to be solved in this area.

In this chapter we do not address the situation of speculative execution - submitting a job to multiple resources and, when one begins to run, canceling the other submissions. We do, however, discuss resource selection (sometimes termed resource discovery [Ber99]), assignment of application tasks to those resources (mapping [Ber99]), and data staging or distribution.

Historically, the most common Grid scheduler is the user, and that is the point of view presented in this chapter. Many efforts are under way, however, to change this situation [Nab99, Zho92, IBM01, GP01, BWF⁺96] and work detailed in Chapters ??, ??, and ?? among others. Many of these are discussed later in this book, but it can be argued that no single system addresses all the needed features yet. In Section 2 we briefly discuss the related Grid Information System interactions expected by a Grid-level scheduler. In Section 3 we describe the three phases a user goes through when scheduling a job over resources on multiple administrative domains—resource discovery, selection, and job execution. Most implemented systems follow a similar pattern of execution. For each step we define the work involved and distinguish it from the work of a common parallel scheduler.

2. GRID INFORMATION SERVICE

The decisions a scheduler makes are only as good as the information provided to it. Many theoretical schedulers assume one has 100 percent of the information needed, at an extremely fine level of detail, and that the information is always correct. In Grid scheduling, this is far from our experience. In general we have only the highest level of information. For example, it may be known that an application needs to run on Linux, will produce output files somewhere between 20 MB and 30 MB, and should take less than three hours but might take as long as five. Or, it may be known that a machine is running Linux and has a file system located at a certain address that ten minutes ago

had 500 MB free, but there is no information about what will be free when one's application runs there.

In general, Grid schedulers get information from a general *Grid Information System* (GIS) that in turn gathers information from individual local resources. Examples of these systems are the Globus Monitoring and Discovery Service (MDS2) [CFFK01, MDS] and the Grid Monitoring Architecture (GMA), developed by the Global Grid Forum performance working group [TAG⁺03], which has several reference implementations under development [pyG, Smi01, CDF⁺01, GLM], and is being deployed as part of the European Data Grid project. These two approaches emphasize different pieces of the monitoring problem although both address it as a whole: MDS2 concentrates on the resource discovery portion, while GMA concentrates on the provision of data, especially streaming data.

While different in architecture, all Grid monitoring systems have common features [ZFS03]. Each deals with organizing sets of sensors (information providers in MDS2 or producers in GMA) in such a way that an outside system can have easy access to the data. They recognize that some data is more statically oriented, such as type of operating system or which file systems are accessible; and this static data is often cached or made more rapidly available. They serve dynamic data in very different ways (streaming versus time-out caches, for example) but recognize the need for a heavier-weight interaction for dealing with data that changes more often. All of these systems are extensible to allow additional monitoring of quantities, as well as higher-level services such as better predictions or quality-of-information metrics [VS03, WSH99, SB99].

Typically, Grid monitoring systems must have an agreed-upon *schema*, or way to describe the attributes of the systems, in order for different systems to understand what the values mean. This is an area of on-going work and research [GLU, DAM, CGS] with considerable debate about how to represent a schema (using LDAP, XML, SQL, CIM, etc.) and what structure should be inherent to the descriptions.

3. STAGES OF GRID SCHEDULING

Grid scheduling involves three main phases: *resource discovery*, which generates a list of potential resources; *information gathering* about those resources and selection of a best set; and *job execution*, which includes file staging and cleanup. These phases, and the steps that make them up, are shown in Figure 1.1.

3.1 Phase 1: Resource Discovery

The first stage in any scheduling interaction involves determining which resources are available to a given user. The resource discovery phase involves

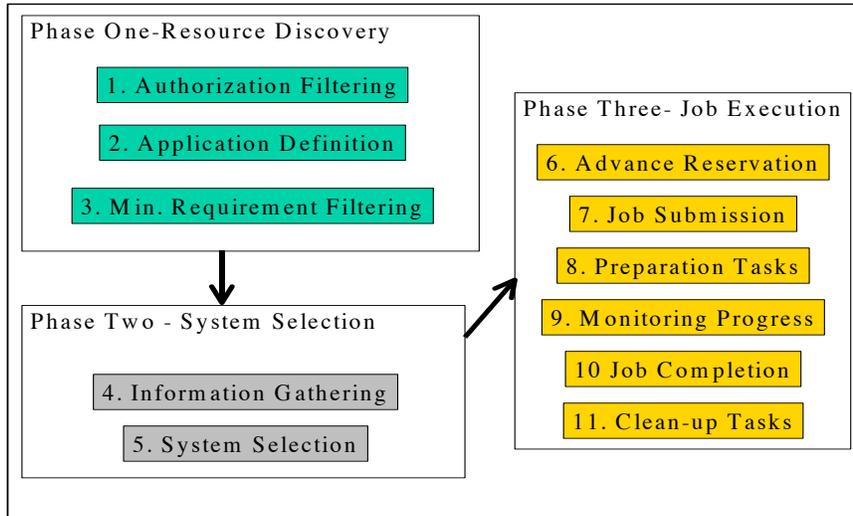


Figure 1.1. Three-phase plan for Grid scheduling.

selecting a set of resources to be investigated in more detail in Phase 2, information gathering. i At the beginning of Phase 1, the potential set of resources is the empty set; at the end of this phase, the potential of resources is some set that has passed a minimal feasibility requirements. The resource discovery phase is done in three steps: authorization filtering, job requirement definition, and filtering to meet the minimal job requirements.

3.1.1 Step 1: Authorization Filtering

The first step of resource discovery for Grid scheduling is to determine the set of resources that the user submitting the job has access to. In this regard, computing over the Grid is no different from remotely submitting a job to a single site: without authorization to run on a resource, the job will not run. At the end of this step the user will have a list of machines or resources to which he or she has access. The main difference that Grid computing lends to this problem is sheer numbers. It is now easier to get access to more resources, although equally difficult to keep track of them. Also, with current GIS implementations, a user can often find out the status of many more machines than where he or she has accounts on. As the number of resources grows, it simply does not make sense to examine those resources that are not authorized for use.

A number of recent efforts have helped users with security once they have accounts, but very little has been done to address the issues of accounting and account management [SN02]. When a user is performing scheduling at the

Grid level, the most common solution to this problem is to simply have a list of account names, machines, and passwords written down somewhere and kept secure. While the information is generally available when needed, this method has problems with fault tolerance and scalability.

3.1.2 Step 2: Application Requirement Definition

To proceed in resource discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources (see Step 3).

The set of possible job requirements can be very broad and will vary significantly between jobs. It may include static details (the operating system or hardware for which a binary of the code is available, or the specific architecture for which the code is best suited) as well as dynamic details (for example, a minimum RAM requirement, connectivity needed, or /tmp space needed). The more details that are included, the better the matching effort can be.

Currently, the user specifies job requirement information as part of the command line or submission script (in PBS [PBS] or LSF [Xu01], for example), or as part of the submitted ClassAd (in approaches using Condor's matchmaking, as detailed in Chapter ??, such as the Cactus work [ADF⁺01] or the EDG broker [GP01]). Many projects have emphasized the need for job requirement information as well, for example with AppLeS [COBW00, BWC⁺03] and the Network Weather Service [Wol98]. It is generally assumed in most system work that the information is simply available.

On a Grid system this situation is complicated by the fact that application requirements will change with respect to the systems they are matched to. For example, depending on the architecture and the algorithm, memory requirements may change, as may libraries needed, or assumptions on available disk space.

Very little work has been done to automatically gather this data, or to store it for future use. This is in part because the information may be hard to discover. Attempts to have users supply this information on the fly has generally resulted in data that has dubious accuracy - for example, notice how almost every parallel scheduler requires an expected execution time, but almost every system administration working with these schedulers compensates for the error in the data, by as much as 50% in some cases.

3.1.3 Step 3: Minimal Requirement Filtering

Given a set of resources to which a user has access and at least a small set of job requirements, the third step in the resource discovery phase is to filter out the resources that do not meet the minimal job requirements. At the end of this

step, the user acting as a Grid scheduler will have a reduced set of resources to investigate in more detail.

Current Grid Information Services are set up to contain both static and dynamic data, described in Section 2. Many of them cache data with an associated time-to-live value to allow quicker response times of long-lived data, including basic resource information such as operating system, available software, and hardware configuration. Since resources are distributed and getting data to make scheduling decisions can be slow, this step uses basic, mostly static, data to evaluate whether a resource meets some basic requirements. This is similar to the discovery stage in a monitoring and discovery service.

A user doing his or her own scheduling will simply go through the list of resources and eliminating the ones that do not meet the job requirements (as much as they are known), for example, ruling out all the non-AFS-accessible resources for applications requiring AFS.

Because the line between static and dynamic data is often one drawn for convenience and not for science, most automatic systems incorporate this feasibility searching into Step 4, where full-fledged queries are made on the system. We maintain that as systems grow, this stage will be an important one for continued scalability of other Grid-level schedulers.

3.2 Phase 2: System Selection

Given a group of possible resources (or a group of possible resource sets), all of which meet the minimum requirements for the job, a single resource (or single resource set) must be selected on which to schedule the job. This selection is generally done in two steps: gathering detailed information and making a decision. We discuss these two steps separately, but they are inherently intertwined, as the decision process depends on the available information.

3.2.1 Step 4: Dynamic Information Gathering

In order to make the best possible job/resource match, detailed dynamic information about the resources is needed. Since this information may vary with respect to the application being scheduled and the resources being examined, no single solution will work in all, or even most, settings. The dynamic information gathering step has two components: what information is available and how the user can get access to it. The information available will vary from site to site, and users currently have two main sources—a GIS, as described in Section 2, and the local resource scheduler. Details on the kind of information a local resource scheduler can supply are given in Chapter ??.

A more recent issue when interacting with multiple administrative domains is the one of local site policies, and the enforcement of these policies. It is becoming common for a site to specify a percentage of the resources (in terms

of capacity, time, or some other metric) to be allocated specifically for Grid use. These details must also be considered as part of the dynamic collection of data.

In general, on the Grid, scalability issues as well as consistency concerns significantly complicate the situation. Not only must more queries be made, but also if resources are inaccessible for a period of time, the system must evaluate what to do about the data needed for those sites. Currently, this situation is generally avoided by assuming that if dynamic data is not available, the resources should be ignored; however, in larger systems, another approach should be considered.

A large body of predictive work exists in this area (for example Chapters ??, ??, and ??) but most of it requires additional information not available on current systems. And even the more applied work [Wol98, GTJ⁺02, SFT98, Dow97] has not been deployed on most current systems.

3.2.2 Step 5: System Selection

With the detailed information gathered in Step 4, the next step is to decide which resource (or set of resources) to use. Various approaches are possible, and we give examples of three in this book, Condor matchmaking in Chapter ??, multi-criteria in Chapter ??, and meta-heuristics in Chapter ??.

3.3 Phase 3: Job Execution

The third phase of Grid scheduling is running a job. This involves a number of steps, few of which have been defined in a uniform way between resources.

3.3.1 Step 6: Advance Reservation (Optional)

In order to make the best use of a given system, part or all of the resources may have to be reserved in advance. Depending on the resource, an advance reservation can be easy or hard to do and may be done with mechanical means or human means. Moreover, the reservations may or may not expire with or without cost.

One issue in having advance reservations become more common is the need for the lower-level resource to support the fundamental services on the native resources. Currently, such support is not implemented for many resources, although as service level agreements become more common (see Chapter ??), this is likely to change.

3.3.2 Step 7: Job Submission

Once resources are chosen, the application can be submitted to the resources. Job submission may be as easy as running a single command or as complicated

as running a series of scripts and may or may not include setup or staging (see Step 8).

In a Grid system, the simple act of submitting a job can be made very complicated by the lack of any standards for job submission. Some systems, such as the Globus GRAM approach [CFK⁺98, GRAb], wrap local scheduling submissions but rely heavily on local-parameter fields. Ongoing efforts in the Global Grid Forum [GGF, SRM] address the need for common APIs [DRM], languages [SD03], and protocols [GRAa], but much work is still to be done.

3.3.3 Step 8: Preparation Tasks

The preparation stage may involve setup, staging, claiming a reservation, or other actions needed to prepare the resource to run the application. One of the first attempts at writing a scheduler to run over multiple machines at NASA was considered unsuccessful because it did not address the need to stage files automatically.

Most often, a user will run scp, ftp or a large file transfer protocol such as GridFTP [ABB⁺02] to ensure that the data files needed are in place. In a Grid setting, authorization issues, such as having different user names at different sites or storage locations, as well as scalability issues, can complicate this process.

3.3.4 Step 9: Monitoring Progress

Depending on the application and its running time, users may monitor the progress of their application and possibly change their mind about where or how it is executing.

Historically, such monitoring is typically done by repetitively querying the resource for status information, but this is changing over time to allow easier access to the data. If a job is not making sufficient progress, it may be rescheduled (i.e., returning to Step 4). Such rescheduling is significantly harder on a Grid system than on a single parallel machine because of the lack of control involved - other jobs may be scheduled and the one of concern pre-empted, possibly without warning or notification. In general, a Grid scheduler may not be able to address this situation. It may be possible to develop additional primitives for interactions between local systems and Grid schedulers to make this behavior more straight-forward.

3.3.5 Step 10: Job Completion

When the job is finished, the user needs to be notified. Often, submission scripts for parallel machines will include an e-mail notification parameter.

For fault-tolerant reasons, however, such notification can prove surprisingly difficult. Moreover, with so many interacting systems one can easily envi-

sion situations in which a completion state cannot be reached. And of course, end-to-end performance monitoring to ensure job completion is a very open research question.

3.3.6 Step 11: Cleanup Tasks

After a job is run, the user may need to retrieve files from that resource in order to do data analysis on the results, remove temporary settings, and so forth. Any of the current systems that do staging (Step 8) also handle cleanup. Users generally do this by hand after a job is run, or by including clean-up information in their job submission scripts.

4. CONCLUSION

This chapter defines the steps a user currently follows to make a scheduling decision across multiple administrative domains. This approach to scheduling on a Grid comprises three main phases: (1) resource discovery, which generates a list of potential resources; (2) information gathering and choosing a best set of resources; and (3) job execution, which includes file staging and cleanup.

While many schedulers have begun to address the needs of a true Grid-level scheduler, none of them currently supports the full range of actions required. Throughout this chapter we have directed attention to complicating factors that must be addressed for the next generation of schedulers to be more successful in a complicated Grid setting.

Acknowledgments

Thanks to the Grid Forum Scheduling Area participants for initial discussions, especially the co-chair, Bill Nitzberg (PBS, Altair Engineering). Many thanks also to Alan Su (UCSD, AppLeS), Dave Jackson (Maui/Silver), Alain Roy (University of Wisconsin, Condor), Dave Angulo (University of Chicago, Cactus), and Jarek Nabrzyski (Poznan, GridLab). This work was supported in part by the Mathematical Information and Computational Sciences Division Subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract W-31-109-Eng-38.

References

- [ABB⁺02] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high-performance computational Grid environments. *Parallel Computing Journal*, 28(5):749–771, 2002.
- [ADF⁺01] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. In *Proceedings of SuperComputing (SC'01)*, 2001.
- [Ber99] F. Berman. High performance schedulers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 12, pages 279–309. Morgan Kaufmann, 1999.
- [BWC⁺03] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the Grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4), April 2003.
- [BWF⁺96] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of SuperComputing (SC'96)*, 1996.
- [CDF⁺01] B. Coghlan, A. Djaoui, S. Fisher, J. Magowan, and M. Oevers. Time, information services and the Grid. In *Proceedings of the British National Conference on Databases*, 2001.
- [CFFK01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, August 2001.

- [CFK⁺98] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Fourth International JSSPP Workshop; LNCS #1459)*, pages 62–82. Springer-Verlag, 1998.
- [CGS] GGF CIM-based Grid Schema Working Group (CGS-WG). <http://www.isi.edu/~flon/cgs-wg/index.htm>.
- [COBW00] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the Grid. In *Proceedings of SuperComputing (SC'00)*, 2000.
- [DAM] GGF Discovery and Monitoring Event Data Working Group (DAMED-WG). <http://www-didc.lbl.gov/damed/>.
- [Dow97] A. Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of the International Parallel Processing Symposium (IPPS)*, 1997.
- [DRM] GGF Distributed Resource Management Application API Working Group (DRMAA-WG). <http://www.drmaa.org/>.
- [GGF] Global Grid Forum (GGF). <http://www.ggf.org>.
- [GLM] GridLab monitoring. <http://www.gridlab.org/WorkPackages/wp-11/index.html>.
- [GLU] GLUE schema. <http://www.hicb.org/glue/glue-schema/schema.htm>.
- [GP01] F. Giacomini and F. Prelz. Definition of architecture, technical plan and evaluation criteria for scheduling, resource management, security and job description. Technical Report DataGrid-01-D1.4-0127-1_0, European DataGrid Project, 2001. Available from http://server11.infn.it/workload-grid/docs/DataGrid-01-D1.4-0127-1_0.doc.
- [GRAa] GGF Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG). <http://www.fz-juelich.de/zam/RD/coop/ggf/graap/graap-wg.html>.
- [GRAb] Globus Resource Allocation Manager (GRAM). <http://www.globus.org/gram/>.

- [GTJ⁺02] D. Gunter, B. Tierney, K. Jackson, J. Lee, and M. Stoufer. Dynamic monitoring of high-performance distributed applications. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, July 2002.
- [IBM01] IBM. Using and administering LoadLeveler for AIX 5L. Technical Report IBM Document #SA22-7881-00, IBM, 2001. Available from <http://publibfp.boulder.ibm.com/epubs/pdf/a2278810.pdf>.
- [MDS] Globus Monitoring and Discovery System (MDS2). <http://www.globus.org/mds>.
- [Nab99] Jarek Nabrzyski. Knowledge-based scheduling method for Globus. In *Proceedings of the 1999 Globus Retreat*, 1999. Also available from <http://www.man.poznan.pl/metacomputing/ai-meta/globusnew/index.htm>.
- [PBS] PBS: The Portable Batch System. <http://www.openpbs.org/>.
- [pyG] Python GMA. <http://sourceforge.net/projects/py-gma/>.
- [SB99] J. Schopf and F. Berman. Stochastic scheduling. In *Proceedings of SuperComputing (SC'99)*, 1999.
- [SD03] GGF Scheduling Disctionary Working Group (SD-WG). <http://www.fz-juelich.de/zam/RD/coop/ggf/sd-wg.html>, 2003.
- [SFT98] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Fourth International JSSPP Workshop; LNCS #1459)*. Springer-Verlag, 1998.
- [Smi01] W. Smith. A framework for control and observation in distributed environments. Technical Report NAS-01-006, NAS NASA Ames, June 2001.
- [SN02] Jennifer M. Schopf and Bill Nitzberg. Grids: The top ten questions. *Scientific Programming, Special Issue on Grid Computing*, 10(2):103–111, August 2002.

- [SRM] Global Grid Forum Scheduling and Resource Management Area (SRM). <http://www.mcs.anl.gov/~jms/ggf-sched>.
- [TAG⁺03] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid Monitoring Architecture. Technical Report GFD-I.7, Global Grid Forum (GGF), 2003.
- [VS03] S. Vazhkudai and J. Schopf. Using regression techniques to predict large data transfers. *Journal of High Performance Computing Applications - Special Issue on Grid Computing: Infrastructure and Application, to appear*, 2003.
- [Wol98] R. Wolski. Dynamically forecasting network performance using the Network Weather Service. *Journal of Cluster Computing*, 1:119–132, January 1998.
- [WSH99] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.
- [Xu01] M. Xu. Effective metacomputing using LSF MultiCluster. In *Proceedings of the First IEEE International Symposium of Cluster Computing and the Grid (CCGrid'01)*, 2001.
- [ZFS03] Xuehai Zhang, Jeffrey Freschl, and Jennifer M. Schopf. A performance study of monitoring and information services for distributed systems. In *Proceedings of the IEEE Twelfth International Symposium on High-Performance Distributed Computing (HPDC-12)*, 2003.
- [Zho92] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proceedings of the Workshop on Cluster Computing*, December 1992.

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory (“Argonne”) under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.