Avian Migration: Condor-Style Job Submission on Standards-Compliant Grid Systems

P. Steele*, M. Saravo*, R. McGrath*, A. Grimshaw
Department of Computer Science
University of Virginia
Charlottesville, VA, USA
Email: pvs5x@virginia.edu, mts5x@virginia.edu, rjm5s@virginia.edu, grimshaw@virginia.edu
**University of Virginia undergraduate students.*

***Abstract*—The Condor system is used worldwide for high-throughput computing, especially scientific computing. Historically, its intuitive submit description language and powerful high-throughput computing capabilities have made it popular. However, with the emergence of synergistic grid standards promising to make grid interoperability easier than ever before, users are presented with a choice: stay with Condor and keep its power but sacrifice interoperability with the increasing number of standards-compliant grids, or switch to the standards-compliant grids, gaining access to a wide array of resources but possibly losing both the simple power of Condor and valuable time and energy in learning how to use a new grid system. This paper aims to present research into creating a third option: enable Condor jobs to be submitted to and run on standards-compliant grids with little to no effort on the part of the user. This software is called Avian.**

***Keywords*-Condor; JSDL; Genesis II; standards.**

I. Introduction

A. Problem

Grid computing is a diverse field that encompasses many different computing paradigms and applications. The specific problem we will address is interoperability between grid systems. Researchers are often presented with circumstances in which it is more expedient for them to run their experiments on different grid systems, but when they encounter systems that are not interoperable, they waste time and effort in attempts to convert from the semantics of one system to another.

The problem is how to increase the utilization of computing resources for research by lowering the time and effort required to run computing jobs on systems that do not share the same job submission syntax and execution semantics. One solution to this problem is for all nonstandard systems to find a way to conform to open standards agreed upon by the community. The motivation for a solution to this problem is a specific need for interoperability between Condor, a high-throughput computing (HTC) system developed at the University of Wisconsin, and Genesis II, a standards-based grid system developed at the University of Virginia (UVA) [1].

B. Importance

The importance of open standards lies in their ability to produce an environment that allows for increased development and utilization of resources through connecting grids which, prior to standards adoption, were not interoperable. Interoperability of grids allows institutions to pool resources without wasting time and energy accounting for differences in proprietary protocols and communication media. The pooling of these resources allows institutions to balance load on their own resources by siphoning off load to other compatible resources, and it enables them to collaborate on even larger scales than previously possible[2, 3].

The value of open standards depends upon adoption by grid computing systems and users migrating toward these systems. With more systems adopting standards and user migration toward these systems, the standards will become firmly entrenched in the grid computing community, increasing the interoperability between future grids as well.

C. Solution

Avian is a step towards standards adoption–it allows users to submit to standards compliant grids using a familiar submission format. Avian translates a job submission file from Condor, a nonstandard but widely used job submission specification, to an open standard, Job Submission Description Language, or JSDL. The reason Condor was chosen was because it is already widely adopted, especially throughout the scientific computing community, which uses grids heavily for their computation needs. JSDL was targeted because it is an open, XML-based, extensible standard being developed by the Open Grid Forum (OGF)[1] [4]. Avian will decrease the effort needed, referred to as the *activation energy*, for the community to move toward standards-based systems.

[1]The OGF is the community of users, developers, and vendors leading global grid computing standardization efforts. The merger between the Global Grid Forum (GGF) and the Enterprise Grid Alliance in 2006 resulted in the OGF; JSDL was originally a GGF standard, as it was created in 2005.

Avian implements key features of Condor to ease the transition; future work on Avian could include extending functionality of standards, including JSDL, to implement further Condor features.

II. Background

A. High Throughput Computing

High throughput computing (HTC) is concerned with the completion of computing tasks over large period of time. HTC environments measure the ability to provide computing over a period of weeks or months [5].

HTC commonly involves executing sequential programs on a number of computing nodes. The environment may consist of a grid of distributed computing resources where computing jobs are matched to available resources. HTC systems are often used by research scientists conducting experiments or calculations that require large amounts of time to finish. These experiments often consist of running one program on several independent sets of data; the lack of any coupling between jobs makes these kinds of experiments ideal for execution in a HTC environment.

B. Condor

Condor is an open-source software framework for HTC developed by the University of Wisconsin [6]. It is used widely in academia, in industry, and by the Open Science Grid (OSG)[2], and is included in the Fedora Linux distribution. One of Condor's strengths is its ability to scavenge cycles, i.e. farming out work to connected idle desktop computers; Condor can also integrate these non-dedicated desktop computers seamlessly with dedicated resources. It also has several different paradigms for running jobs, called *universes*[3]. Each universe is designed to satisfy a set of needs for a job environment.

Arguably its biggest strength, however, is how Condor jobs are described, which is referred to in Condor as the *submit description language*. The submit description language includes multiple submission from one file, workflow management through its Directed Acyclic Graph Manager (DAGMan), resource-based matchmaking, and other advanced features targeted to HTC.

1) DAGMan: DAGMan is Condor's workflow management system [6]. It represents dependencies between jobs as a directed acyclic graph; individual nodes represent DAGMan jobs, and edges between nodes represent dependencies. It is important to note that one DAGMan job is not limited to one instance of a program submitted to run under Condor; a Condor submit description file allows for multiple submissions from the same file, and each DAGMan job in the DAG input file (the input file used by DAGMan) is associated with one Condor submit description file.

DAGMan is a key feature that provides even more flexibility to Condor. Besides being an intuitive way to describe job dependencies, it allows for the same Condor submit description file to be used for multiple different DAGMan jobs through the use of macros. Macros are defined in a DAG input file for each job defined in that file; each macro has a name and value, and whenever the name of the macro is seen in the corresponding DAGMan job's submit description file, it is replaced with the macro's value.

C. JSDL

JSDL is an open XML standard from the OGF [4]. In the context of grid computing, it allows specifically for describing jobs: job name, job description, resource requirements (e.g. total RAM available, CPU clock speed), execution limits (e.g. maximum wallclock time, maximum memory to be consumed), file staging, and the command to execute (including any arguments, environment variables, input/output redirection, etc.).

In addition to the explicitly defined abilities of JSDL, the fact that JSDL is XML makes it extensible. An example of this extensibility is the Parameter Sweep extension, which allows for definition of multiple jobs in one JSDL file, as long as the value of one or more job parameters changes in some preordained fashion from one job to the next [7]. This extension was specified in May 2009, four years after the original JSDL 1.0 specification; it is slowly being implemented in standards-compliant grid systems. The extensibility of JSDL is mitigated somewhat by the resistance of the user community to change, as evidenced by the Parameter Sweep extension only recently being adopted and worked into standards-based grids.

While one of JSDL's strengths is its extensibility, one of its major weaknesses is its lack of a schedule description language. As opposed to how DAGMan handles Condor jobs, dependencies between JSDL jobs must be managed by users. There are also other differences between JSDL and the Condor submit description language. JSDL, being a standard born out of a standards organization, was subject to much discussion about what would go into the specification, and as such is the minimum agreed-upon standard. Condor, on the other hand, is a focused but nonstandard effort of describing jobs; the only entity vetting new Condor functionality is the Condor team itself, which makes it easier to agree on new functionality.

D. BES

A Basic Execution Service is a service that accepts requests from clients to initiate, monitor, and manage computation [8]. In order to run on a BES, JSDL jobs must match their requirements to the BES's BES-Factory

[2]The Open Science Grid is a worldwide grid, which is used by scientists and researchers for problems which are too computationally-intensive for any one supercomputer or data center. It is administered by the Open Science Grid Consortium.

[3]More information about Condor universes can be found in the most recent manual [6], or at http://www.cs.wisc.edu/condor/manual/v6.4/2_4Road_map_Running.html

Attributes[4]. The BES-Factory port-type provides a Web-Service-accessible characterization of the resources the BES makes available to activities.

E. RNS

The Resource Namespace Service (RNS) is a standard, proposed for the first time in 2006, that describes a simple way of mapping names to endpoints within a grid [9]. Conceptually, it supports a three-tiered naming architecture of *human interface names, logical references*, and *endpoint references*, which enables the construction of a global, uniform, hierarchical namespace. This namespace is useful because it provides a familiar way to access the rapidly-growing number of service resources (e.g. Web services and their corresponding applications) that are capable of being referenced in a grid environment.

F. Genesis II

Genesis II is a an open source grid system based on JSDL, BES, RNS, and other standards; it is the first such standards-based grid [1]. Its purpose is twofold: it provides a production-caliber grid environment for scientific research at the University of Virginia, and it serves as a test bed for grid research. Its flexibility and availability make it ideal to develop and evaluate new grid technologies and models. As part of this second purpose, it serves as a testing area to validate the standards proposed by the OGF and others working toward interoperability through community standards.

III. Solution

A. Conceptual Solution

At an abstract level, Avian acts as a user agent by accepting the user's Condor file and then working with the grid to execute the job as specified. The intended user scenario is for users to open Avian on their personal computer and have all of their submission files and input data located there. Users craft the necessary submit description and DAG input files and then run a compatibility checker to see if unsupported commands are used. They specify certain preferences concerning the location of their home directory and the path to the grid shell utility. Users will then direct Avian to execute the job. Avian informs users of the status of the job and stages their data to and from the grid as appropriate. Once the job is complete, users exit the program and can inspect the output of their job.

Within Avian, once users submit the job, the transformation engine transforms Condor submit description files to JSDL files. The necessary data is then staged out to users' home directories on the grid. Avian submits the JSDL files and determines their status by periodic polling. If a DAG input file is submitted, Avian will launch new jobs as the dependencies in the DAG are fulfilled. Once all jobs are complete, Avian stages out the resulting data from the users' home directories on the grid to their local computers.

[4]The BES-Factory Web-Service port-type allows clients to create, monitor, and control sets of activities, and monitor BES attributes.

```
Executable     = foo
Universe       = vanilla

Error   = err.$(Process)
Input   = in.$(Process)
Output  = out.$(Process)
Log     = foo.log

Initialdir = dir_$(Process)
Queue 10
```

Figure 1. An example Condor file demonstrating some advanced features of Condor (adapted from Example 6 and Example 7 on http://www.cs.wisc.edu/condor/quick-start.html.).

B. Applied Solution

Since Avian supports a subset of the available Condor and DAGMan tags, a compatibility checker was created. This checker warns users about possible inconsistencies between what submit description files demand and what Avian supports. Tags that are flagged may be ignored, or cause the program to act in an unexpected manner.

In order to run jobs on Genesis II, they must be described in JSDL. Avian includes a transformation engine, created to convert Condor submission files into proper JSDL files. This involves parsing the Condor directives and determining the relevant commands that can be translated into a JSDL equivalent. Avian processes relevant commands into attributes of an object that represents a JSDL job, and writes the JSDL job to an XML file which conforms to the JSDL standard.

The parser within the transformation engine implements advanced features of Condor. An example Condor file with several of these advanced features is shown in Figure 1.

The queue command directs the system to run the job the specified number of times. Avian creates separate JSDL jobs for the specified number of queue commands. Differentiating between jobs and their related files is made easy by Condor's process macro substitution, which Avian duplicates. Wherever Condor encounters "$(process)", it substitutes the Condor process ID, which is an integer that starts at 0 and is incremented by 1 for every subsequent job that Condor launches from the same submit description file [6]. Avian keeps track of Condor process ID's on the client side program in order to create JSDL files for each job it needs to launch.

For the job in Figure 1, this would translate to ten jobs being launched, each with their staging data in different directories. The resulting jobs would expect in.0, in.1, in.2, etc., up to in.9, to be located in dir_0, dir_1, dir_2, etc., up

to dir_9, respectively. When the jobs finish, out.0 and err.0 would be located in dir_0, out.1 and err.1 in dir_1, etc.

Once Avian creates the appropriate submission files, it interacts with the target grid, Genesis II. Genesis II provides a command line utility with a set of tools that mimic standard UNIX tools, such as cat or cp. The utility also enables access to grid specific tools, such as qsub, which submits a job to a queue, or qstat, which returns the status of a job. Avian accesses these tools by spawning a Windows command line process in the background which runs the grid utility. The utility and Avian interact via input and output streams. These allow Avian to write commands and read their output. When Avian submits a job, the grid utility is passed a string that contains the command, the queue to submit to, and path to the file on the local computer.

In addition to submitting jobs, Avian frequently needs to check the status of a job. In this circumstance, Avian passes the appropriate grid command to the utility and parses the output from the buffered reader. This approach allows development with the grid to proceed at a high level and does not re-develop pre-existing functionality. Another advantage is that Avian, being loosely coupled with the grid by design, was able to weather a recent revision of Genesis II with only minor difficulties.

C. DAGMan Emulation

Avian emulates DAGMan functionality by enforcing the dependencies of the submitted jobs locally and submitting new jobs only when dependent jobs have completed. The first step is to parse the DAG input file which describes the DAGman graph using the syntax "PARENT [job] CHILD [child]". From these dependencies a node object is created within Avian consisting of the converted JSDL job and its children.

If the Condor submission file specifies multiple queue commands, the resulting JSDL files are associated with the node which subject these jobs to the dependencies from the DAGMan graph. Avian then determines which nodes are not children of any other node; these nodes are able to run initially. When these jobs complete, Avian then returns a list of jobs that are able to run. The workflow dependencies of the DAGMan function are implemented using a local application and JSDL submission files.

Avian handles data staging by copying data to the grid and informing the BES container where to access the data on the grid. These files are located in the user's computer's local directory with the Condor submit file or a in a relative subdirectory specified by the "initialdir" command.

When Avian stages data in, a temporary directory is created within the user's home directory on the grid. The data is copied to this location using the grid command line tool "cp". Once the data is on the grid, the JSDL DataStaging tag is created to reflect its location. A RNS path to the data is inserted into the Source tag within the DataStaging element. Once the job completes, the grid reads the RNS path in the Target tag and writes the output files to the specified location under the user's grid home directory. The data is then copied back to the user's computer.

Since Avian must be running to determine when jobs finish and when to start new jobs, jobs that run for an extended period of time could behave unexpectedly if the user terminates Avian. To remedy this situation, Avian saves its state so that the current set of jobs can be resumed later. Avian's state is stored in a serializable object that is written to disk. When the user loads the saved execution run, Avian knows which jobs were submitted, which jobs still need to run, and what data needs to be staged out. The user also has the option of aborting the run entirely without saving state. When all jobs complete, or the run is aborted, Avian can clean up data left on the grid and terminate jobs in the queue.

IV. Results

Avian was able to complete a number example jobs provided by the Condor project website. This included simple Condor submit files (a modified one is shown in Figure 2), Condor submit files with basic macros, and Condor submit files scheduled through a DAGMan file.

As shown in Figure 2, Avian can translate basic Condor commands, including specifying an executable, arguments to that executable, and environment variables to transfer to the grid environment. Important to note is that in the Condor submit description file (the top part of Figure 2), the arguments and environment variables are specified in the new syntax, but Avian also accepts the old syntax for specifying arguments and environment variables. However, due to ambiguity in the old syntax, the Condor team suggests using the new syntax [6], as does the Avian team.

Also shown in Figure 2's translation is Avian's ability to parse, to an extent, Requirements ClassAds[5]. The Condor file specifies that the job requires an Intel x86 64-bit processor architecture, and this is reflected in the JSDL beneath it. Of course, no real "Hello World" application would need to be run on a 64-bit processor; this line was simply put into the submit description file to demonstrate Avian's capability to recognize Condor's processor architecture specifications and supply the correct JSDL enumeration value to the JSDL file. Avian can also do this for Condor's operating system specifications.

In addition to the above example, the translation engine was able to handle a more complicated research job from UVA's biomedical engineering department. This job, shown in Figure 3, is launched by a DAG input file which iterates across the values of macros i and j from 1 to 14 and 1 to 1000, respectively, launching one job for each (i, j) pair.

[5]ClassAds are a mechanism that Condor uses to perform matchmaking. For more information, consult the most recent Condor manual [6], or http://www.cs.wisc.edu/condor/manual/v6.4/4_1Condor_s_ClassAd.html.

```
Executable = helloworld.bat
Universe = vanilla
requirements = arch == "x86_64"
arguments = "'spacey ''quoted'' argument' one ""two"""
getenv = false
environment = "one=1 three='spacey ''quoted'' value' two=""2"""
coresize = 20

Initialdir     = run_1
Queue
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<jsdl:JobDefinition xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl" xmlns="http://www.e
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <jsdl:JobDescription>
  - <jsdl:Application>
      <jsdl:ApplicationName>helloworld.bat</jsdl:ApplicationName>
    - <jsdl-posix:POSIXApplication>
        <jsdl-posix:Executable>helloworld.bat</jsdl-posix:Executable>
        <jsdl-posix:Argument>spacey 'quoted' argument</jsdl-posix:Argument>
        <jsdl-posix:Argument>one</jsdl-posix:Argument>
        <jsdl-posix:Argument>"two"</jsdl-posix:Argument>
        <jsdl-posix:Environment name="two">"2"</jsdl-posix:Environment>
        <jsdl-posix:Environment name="one">1</jsdl-posix:Environment>
        <jsdl-posix:Environment name="three">spacey 'quoted' value</jsdl-posix:Environment>
        <jsdl-posix:CoreDumpLimit>20</jsdl-posix:CoreDumpLimit>
        <jsdl-posix:WorkingDirectory>run_1</jsdl-posix:WorkingDirectory>
      </jsdl-posix:POSIXApplication>
    - <jsdl:Resources>
      - <jsdl:CPUArchitecture>
          <jsdl:CPUArchitectureName>x86_64</jsdl:CPUArchitectureName>
        </jsdl:CPUArchitecture>
      </jsdl:Resources>
    </jsdl:Application>
  - <jsdl:DataStaging>
      <jsdl:FileName>helloworld.bat</jsdl:FileName>
      <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
    - <jsdl:Source>
        <jsdl:URI>http://people.virginia.edu/~pvs5x/stagingDir/helloworld.bat</jsdl:URI>
      </jsdl:Source>
    </jsdl:DataStaging>
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

Figure 2. A simple Condor job, demonstrating a few simple direct translations from Condor to JSDL. The "initialdir" and "queue" commands are not fully translated because they are handled programmatically in Avian.

```
universe   =  vanilla
executable =  /unsup/R/bin/R
getenv     =  True
environment = i=$(i); j=$(j)
+RJob = TRUE
requirements = ( preferR == TRUE ) && ( memory >= 256 )
rank = kflops
arguments  = --vanilla --no-readline
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files =
Source.R,Source-coxen.R,rnaa,mutt,cnvv,senss,SENS.RData,RNA
,sen.res.X5,sen.res.X6,sen.res.X7,sen.res.X8,sen.res.X9,sen
l.txt
input      =  sens-rna-mut-sim.R
output     =  output
error      =  sim.err
log        =  sim.log
notification = never
queue 1
coresize   =  0
```

Figure 3. A more complicated Condor job, which is used by the UVA biomedical engineering department.

When Avian parses the DAG input file, it creates a new JSDL file for each one of these jobs with the correct macro substitutions. One feature of this Condor submit description file that was not implemented was the Rank feature. Rank allows Condor users to specify a preference ranking for resources in terms of a mathematical formula, the elements of which are attributes of the resource. Ranking resources for preference is not supported in JSDL at this point; it is possible that in the future, Avian can emulate this functionality with a client side application accessing BES container attributes. However, for the job to run, Rank implementation is not necessary; the job simply needs to find *a* resource or set of resouces, not the most optimal one.

A. Limitations

1) Scope of Condor Support: One limiting factor of Avian is the scope of support for Condor commands and functionality. The initial motivation for Avian was to be a proof of concept; the amount of supported commands reflects this motivation. There were two types of limitations that determined which functionality made it into this iteration of Avian: fundamental and implementation. Fundamental limitations were ones for which there was a semantic gap between Condor and JSDL that could not be bridged. Implementation limitations were caused by lower priority functionality not being a part of Avian's proof-of-concept plan. Decisions on priority of different functionality were made by examining example files on the Condor webpage, tutorials, HTC bootcamp materials, and examples from researchers who use Condor. Additionally, Condor commands were only supported if there was a matching JSDL tag. Tables 1, 2, and 3 summarize Avian's current coverage of Condor submit description language, including why we chose to leave certain commands out of the proof-of-concept plan and the fundamental limitations we encountered.

In general, Avian follows the Condor grid universe semantics [6] as closely as possible, as it is intended for use in a grid environment.

Condor includes a large array of commands omitted from Tables 1, 2, or 3. Some of these are advanced commands that are not a one-to-one mapping, and would have required emulation programs to be implemented. The rest of the commands were specific to other Condor universes; the intended scope of Avian is currently just the vanilla universe commands. The vanilla universe is the subset of all the other universes' commands and, as such, was a logical scope for Avian as a proof of concept.

2) Scope of DAGMan Support: The extent of Avian's support for DAGMan is described in Table 4.

We do not support the DATA command because it specifies a job to be managed by the Stork data placement server[6]. DIR is an option for JOB which specifies a working directory for the DAG node; as the DAG node is emulated to run on the client side and submit Condor jobs from within it to Genesis II, DIR is not necessary for proper function. SCRIPT specifies pre- or post-processing scripts, which are run before or after the specified job executes, respectively. Scripts are optional for each job, and in all the real DAG input files we worked with, there were no SCRIPT keywords. Additionally, pre- and post-processing scripts can be run by the user individually before and after their jobs have finished. Therefore, Avian does not support SCRIPT functionality as of this iteration.

This scope of coverage appears sparse at first glance. However, the supported commands are the ones used in the majority of circumstances we were able to examine, while unsupported commands in some cases represent esoteric features seemingly used only by a minority of users. The present limitations of the JSDL standard preclude many commands that allow the user to modify certain other options. While the current functionality of Condor allows greater options and modifications than JSDL, working within the JSDL standard shows that Avian covers a useable set of commands for HTC.

3) Other Limitations: We do not support the full set of Condor tags; rather, we support the ones we believe will provide the most coverage for the most users. To simplify this end, we borrow the semantics from the grid universe, but cover mostly the basic, i.e. vanilla universe, commands. Data staging is done with the Condor directives, transfer_input_files and transfer_output_files, which is not always the case in real Condor files; if Condor files are not specified to run in the grid universe and transfer_output_files is not specified (a common occurrence), Condor will automatically back all files in the job's temporary working directory which have been modified or created by the job [6].

V. Conclusion

A. Remarks

This paper presents a valid proof of concept to address the issue of providing a solution to support running Condor jobs in a standards-based environment. Avian represents the core functionality of an application that would allow a Condor user to work in a standards-based environment. However, there are still limitations with Avian, specifically the extent of its support for Condor functionality and the current supported capabilities of the related standards.

Avian achieved a number of the original design goals of the project. It effectively parses a usable subset of Condor commands to JSDL while implementing features that aid in submitting HTC. It integrates with the Genesis II Grid to

[6]Stork is a batch scheduler which specializes in data placement and movement; more information can be found at the Stork project home page, http://www.cct.lsu.edu/ kosar/stork/index.php.

Table I
CONDOR BASIC AND MISCELLANEOUS COMMAND COVERAGE.

Command	Implemented?	Notes
arguments = <argument_list>	Yes	-
environment = <parameter_list>	Yes	-
error = <pathname>	Yes	-
executable = <pathname>	Yes	-
getenv = <True | False>	Yes	-
input = <pathname>	Yes	-
log = <pathname>	Yes	-
log_xml = <True | False>	No	If True, the log is written in ClassAd XML; this option was not found in any of the example Condor files we examined, nor is it necessary for the proper function of any application.
notification = <Always | Complete | Error | Never>	No	The only time we saw this was specifying Never.
notify_user = <email-address>	No	Also not absolutely necessary for jobs to run in the grid.
output = <pathname>	Yes	-
priority = <integer>	No	JSDL does not provide a way to enforce priority across jobs. However, Genesis II does; this functionality could be implemented in a future version of Avian.
queue [number-of-procs]	Yes	-
universe = <vanilla | standard | scheduler | local | grid | mpi | java | vm>	Not complete	The scope of Avian was intended to be the vanilla universe commands; the commands in the vanilla universe are the subset of all other universe's commands and are usable in the other universes.
coresize = <size>	Yes	Specifies the maximum size, in bytes, of the core file, if one is produced. Core files larger than this amount will not be retained.

Table II
CONDOR MATCHMAKING COMMANDS COVERAGE.

Command	Implemented?	Notes
rank = <ClassAd Float Expression>	No	JSDL does not support ranking eligible resources for a job to run on based on preference. This is a fundamental limitation.
requirements = <ClassAd Boolean Expression>	Not Complete	More complicated Boolean expression parsing was not implemented; OR is not supported in JSDL. This is a fundamental limitation.

Table III
CONDOR FILE TRANSFER COMMAND COVERAGE.

Command	Implemented?	Notes
should_transfer_files = <YES | NO | IF_NEEDED >	No	We will transfer all files regardless of whether this is specified, to make sure the user does not miss files that they need.
transfer_executable = <True | False>	No	See above.
stream_error = <True | False>	No	There is no analog in JSDL for redirecting standard streams back to the host in real time. This is a fundamental limitation.
stream_input = <True | False>	No	See above.
stream_output = <True | False>	No	See above.
transfer_input_files = <file1,file2,file...>	Yes	-
transfer_output_files = <file1,file2,file...>	Yes	JSDL requires that the user specify which files are staged out.
transfer_output_remaps = < " name = newname ; name2 = newname2 ... ">	No	This command specifies the name (and optionally path) to use when receiving staged out files from the completed job. Not supported in JSDL. This is a fundamental limitation.
when_to_transfer_output = <ON_EXIT|ON_EXIT_OR_EVICT>	No	There is no analog in JSDL for this command. This is a fundamental limitation.

stage data, submit jobs, and kill jobs and check job status. Avian allows for dependency based scheduling and shows the feasibility of constructing such a feature based on top of JSDL.

These achievements show that creating a standards based system can deliver functionality that users expect from HTC systems. With effort, systems that build on agreed-upon standards can be as useful as those that do not and provide the advantages of open standards.

B. Future Work

There are notable limitations with the current state of Avian that could be addressed in future efforts. This proof of concept would benefit from increased support for a number

Table IV
DAGMAN COMMAND COVERAGE.

Command	Implemented?
PARENT/CHILD	Yes
JOB	Yes
VARS	Yes
DATA	No
DIR	No
SCRIPT	No

Condor commands that have a clear JSDL counterpart. Advanced Condor features, such as support for including a graph within another graph, or generating a new graph of jobs to complete based on a partial failure, could be emulated and supported on standards-based systems. Additional features could be added to Avian if it is modified to interface with different grid management systems; to ease development of these modifications, the targeted grid management systems would ideally have an interactive, command-line-based method of interaction. Avian would also benefit from being able to run on multiple client operating systems. Currently, Avian only runs on Windows.

Resource matchmaking is minimally supported in JSDL; the extent to which Avian supports Condor resource matchmaking roughly matches that of its support in JSDL. Ranking resources is not supported in JSDL, but could possibly be implemented in another way in the future. Also, while the vanilla universe is very useful in and of itself, the other Condor universes also have value. Future research could also explore how best to emulate these universes using Avian.

Present capabilities of Avian to handle jobs that are based on the same submit description file, but have a different set of parameters for each, could be more efficiently implemented using JSDL's ParameterSweep extension. Functionality for this extension has recently been added to Genesis II; future research could explore the benefits and drawbacks (if any) of using ParameterSweep instead of creating individual JSDL files.

Lastly, future work could enhance the capabilities of DAGMan, including supporting more of the functionality detailed in Table 4, supporting DAG input files as nodes inside other DAG input files, and implementing recovery for failed DAG nodes.

One continued question throughout the development of Avian would be its value in a real-world environment. A continuation of this work could involve a trial with a substantial number of experienced Condor users using this application to work on a standards based system. This would show the practicability of this approach for migrating users toward standards based systems.

REFERENCES

[1] M. Morgan and A. Grimshaw. "Genesis II-standards based grid computing," in *Seventh IEEE International Symposium on Cluster Computing and the Grid*, 2007, p. 611

[2] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. "The Open Grid Services Architecture, Version 1.0," *Open Grid Forum*, January 2005. [Online]. Available: http://www.gridforum.org/documents/GFD.30.pdf.

[3] H. Kishimoto, J. Treadwell. "Defining the Grid: A Roadmap for OGSA Standards Version 1.0," *Open Grid Forum*, September 2005. [Online]. Available: http://www.ggf.org/documents/GFD.53.pdf

[4] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, "Job Submission Description Language (JSDL) Specification V1.0," *Open Grid Forum*, November 2005. [Online]. Available: www.gridforum.org/documents/GFD.56.pdf.

[5] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for high throughput computing," *SPEEDUP*, vol. 11, 1997.

[6] M. Livny, M. Solomon, B. Burnett, D. Bradley, T. Cartwright, P. Couvares, et al. "Condor Version 7.4.1 Manual," *Condor Manuals*, April 2010. [Online]. Available: http://www.cs.wisc.edu/condor/manual/v7.4/condor-V7_4_2-Manual.pdf [Accessed: April 29, 2010]

[7] M. Drescher, A. Anjomshoaa, G. Williams, D. Meredith. "JSDL Parameter Sweep Job Extension," *Open Grid Forum*, May 2009. [Online]. Available: www.gridforum.org/documents/GFD.149.pdf.

[8] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, M. Theimer. "OGSA Basic Execution Service Version 1.0," Open Grid Forum, November 2008. [Online]. Available: http://www.ogf.org/documents/GFD.108.pdf.

[9] M. Pereira, O. Tatebe, L. Luan, T. Anderson. "Resource Namespace Service Specification," Open Grid Forum, May 2006. [Online]. Available: http://www.ggf.org/GGF17/materials/272/Resource_Namespace_Service_Refactored.pdf.