BioHPC

Computational Biology Application Suite

for High Performance Computing

Administrator's guide

This document is under construction

Last updated: 2/6/2009

For questions and comments contact: cbsu@tc.cornell.edu

Contents

1 Overview: BioHPC from users' perspective

Low cost of PC clusters makes them an attractive high-performance computing (HPC) solution in many fields of science and business. However, effective utilization of computational power offered by such clusters requires substantial technical and system programming knowledge, which is often a major obstacle to an average user – typically an expert in his/her own field, but not necessarily in computer programming or intricacies of cluster job schedulers.

The objective of the BioHPC is to simplify access to resources of computer clusters. The prelude to understanding of the components of the system is to look at a job submission process from the user's perspective. Instead of dealing with the actual cluster (or multiple clusters), batch scripts and job scheduler, all the user has to do to submit a job and retrieve results is to interact with an easy-to-use web interface. First, the user selects the application he/she wants to use. BioHPC offers a variety of bioinformatics and computational biology applications, including parallelized BLAST, HMMER, InterProScan, and quite a few population genetics programs. However, the structure of BioHPC is open, so that arbitrary applications, not necessarily related to computational biology, may be easily added (see subsequent sections). For each application, the interface features a custom submission web page – essentially a web form collecting all the information needed to submit a job with this application, such as input files to be uploaded, command line options, or, in the case of parallel applications, number of processors to execute on. The user is also asked to provide a valid e-mail address where all information about the job will be automatically sent. Optionally, if multiple compute clusters are available, the user can also select the specific cluster where his job should run. A simple alternative is to simply rely on a meta-scheduler built into BioHPC. After providing the necessary information, the user clicks the "Submit" button, and shortly afterwards he receives an automated notification email informing him that his job has been submitted. Two more emails will be sent later on: when the job starts executing, and when it finishes for any reason. All the notification email messages contain http and ftp links to all the relevant output files, some of them available during the run (so that the job progress monitoring is possible), and some (final output files) available only after completion of the job. Links are also provided which make it possible for a user to cancel a job if need be or to delete al the job's data from the BioHPC servers.

Being a web interface, BioHPC is able to serve any user equipped with a web browser and a valid e-mail address – no accounts or passwords are necessary. There is, however, a possibility to create a pool of the so-called "registered users" who will enjoy special privileges while working with BioHPC. Such users are issued a password which they can use to log in to the interface prior to submitting the jobs. While logged in, they may be entitled to submit more jobs with longer timeouts then other users and they will have access to a handy job viewing tool, more convenient than notification emails. Access to some applications, e.g. the ones consuming a lot of resources, may even be restricted to registered users only. Registered users are also allowed to store their private databases (such as those used with BLAST) permanently on the server. Finally, a subset of registered users may be given administrative privileges. Administrators are responsible for configuring users' privileges, cluster and application settings, and – perhaps most importantly – for monitoring the jobs, ensuring their undisturbed flow through the clusters, detecting and fixing hardware or software problems which may disrupt this flow.

BioHPC values user's privacy. The job data generated by one user cannot be accessed by other users except administrators. The employed security measures, although not completely watertight, proved entirely sufficient for research purposes served by the existing BioHPC installation.

Some of the BioHPC data-mining applications, such as BLAST, HMMER, or InterProScan, require large files to be replicated on local disks of each node of the cluster(s). Because of their functionality, these files are usually referred to as "databases", although they are not really databases in a sense of any commonly used database engine, such as Microsoft SQL server, for example. This distinction is important for further discussion, since a genuine MS SQL database is also utilized by BioHPC, for the purpose of storing various parameters and job information. An important aspect of BioHPC, generally hidden from a user, is application database administration, i.e., update and distribution of database files on the nodes of the cluster(s).

2 In depth: BioHPC from administrator's perspective

The BioHPC suite may be considered from two different points of view: structural and functuional. Structurally, BioHPC is a suite of programs running on machines which have to satisfy certain hardware and software prerequisites. These structural aspects of the suite are discussed in the Section 2.1. This will be followed by a detailed description of functional aspects of BioHPC.

2.1 Structural components

Simply speaking, BioHPC is a collection of **web pages** implemented in **ASP.NET** (where ASP stands for Active Server Pages), **application** executables [the actual programs submitted to run on the cluster(s)], and various **auxiliary programs** called by the web pages or running periodically as scheduled tasks, usually on the web server. In the spirit of ASP.NET, each web page has two basic components. The HTML component is what is displayed in the browser on user's local machine. If needed, javascript procedures embedded in the HTML document allow for dynamic changes on the page based on the user's inputs. The second component is the "**code behind**" – a collection of procedures (written in C#) that run on the **web server** when activated from a web page, usually by clicking on an appropriate button.

Various parameters and settings describing users, clusters, and applications are stored in an MS SQL database (typically called `CBSUJOBS`), running on an **SQL server**, as well as in the `web.config` file which determines the configuration of BioHPC web pages. The "code behind" the web pages contacts the database and `web.config` on the regular basis. Most importantly, the `CBSUJOBS` database contains information about each and every job submitted to the clusters. This information, stored in records of the table `jobstatus`, is dynamically updated throughout the lifetime of a job by various programs form the suite (to be discussed later). The most important entries in `jobstatus` are the job ID (a unique integer number incremented by 1 each time a new job is registered), job control number (a large signed randomly generated integer, which serves as a password for access to the files generated by the job and its database entries – see Section 2.2.2 for more details), and a job status flag (an integer representing the current status of the job, such as "queued", "running", "finished", etc. – see Section 2.2.4 for more details). This flag will be referred to as **database status** of the job.

All nodes of compute clusters served by the BioHPC suite (except for the JSDL clusters – to be addressed later) must have access to a **common network drive**, located on the **file server**. On the **web server**, this drive must be seen as `H:` and the BioHPC-related files are stored in `H:\CBSU`. In particular, the subdirectory `H:\CBSU\JOBS` holds job data input, output, and script files (each job has its own subdirectory named after its job ID, for example, `H:\CBSU\JOBS\12345`). Subdirectories containing application executables and auxiliary programs are also located in `H:\CBSU`. The exact names of these subdirectories are configurable either in the `CBSUJOBS` database or in `web.config` file (see Sections 3 and 4 for more info). Jobs running on compute nodes access these directories on a regular basis. A special perl driver mounting the `H:` drive on the web server at all times is a part of the BioHPC installation. On compute clusters, the shared network drive does not have to be seen as `H:` (i.e., it does not have to be explicitly mounted as such) - the UNC path can be specified instead (see Section 4.3). BioHPC will automatically assign the letter `H` to the UNC path through the Windows net use command.

Upon the completion of a job, its job directory on the file server can, optionally, be replicated on another machine acting as an **ftp server**. From this machine, data can be retrieved via the ftp protocol as opposed to http used to get the data off the file server directories (the notification e-mails contain both the http and ftp links to the job's output files). This is especially important if the output files are large and not suitable for http transfer. Besides, backing up job data on an ftp server increases storage reliability. The ftp functionality can be configured in BioHPC if an ftp server is available.

The discussion above highlights the hardware and software requirements of BioHPC suite, summarized here:

- At least one **compute cluster** running Windows Server 2003 with Compute Cluster Server (CCS) pack or Windows HPC Server 2008 with the native MPI library and scheduler. As an alternative to CCS, BioHPC also supports the Velocity scheduler with MPIPRO library, as well as JSDL job submission protocol, so that remote clusters can also be served. More on remote clusters – in Section 6.
- A **web server** running **IIS**, **Microsoft Windows SDK**, **Microsoft .NET 3.0 Runtime** to host the web component of BioHPC. If changes in the source code are anticipated, **MS Visual Studio 2005**has to be available on at least one machine able to connect to the web server and modify/recompile the BioHPC solution there.
- A **file server** with a **network drive** accessible from all the nodes of all clusters (except for remote clusters accessed via JSDL); on the web server this drive must be mounted as `H:`.
- An **ftp server** (optional).
- An **MS SQL server** to host the `CBSUJOBS` database. The free version, MS SQL Server Express Advanced is sufficient.
- **Interface Domain User (IDU)** – a Windows domain account (name is configurable) serving as the owner (in Windows sense) of all files and processes generated by BioHPC. The access rights for IDU must allow submitting jobs to clusters, accessing files on network drive and others (specified in detail in installation instructions).

In the original implementation of BioHPC, all the servers mentioned above are running on separate machines. However, there is no reason why a single machine, such as the cluster head node, could not be used to run all the needed services. Such a solution is entirely feasible, especially for small-size clusters.

See http://biohpc.com/inst.aspx for more detailed instructions on how to install and configure the hardware and software components of BioHPC.

2.2 Functional components

The web pages and auxiliary programs comprising BioHPC can be roughly grouped into a few classes, each class providing a well defined function. Some of the web pages, most important from the users' point of view, serve as job submission interfaces for various applications. The purpose of others is to provide access to the jobs' output files and other resources, and to provide accurate information about the status of the jobs. A large subset of web pages and programs is dedicated to the administration of BioHPC system. The next few subsections deal with general aspects of job submission, file access, job control, and status monitoring. Discussion of these core functional components of BioHPC will set the stage for the description of administrative tools, where all the nitty-gritty details will be discussed further.

2.2.1 Job submission

Although both the HTML portion of a submission page and the associated "code behind" are application-specific, they follow the general scheme to be outlined here.

When the page is presented to the user, a routine `Page_Load` is run on the server, which contacts the SQL database to read off basic information about this user, clusters, and application programs he/she is allowed to run. This information is used to initialize some of the controls on the page. Once a user fills out a job submission form and clicks "Submit", another procedure, usually called `Submit_Click`, is executed on the server. This procedure collects and validates the entered data (for example, the existence and format of the uploaded input files may be verified, correctness of the command line may be checked, etc.), and, upon successful validation, it registers a job, i.e., creates a new record in a table (called `jobstatus`) in the **MS SQL** database called `CBSUJOBS`. This record contains complete information about the job, including the unique job ID (automatically generated by the database engine), path of the job directory (still to be created) where all the job files will be stored, ID and e-mail address of a user who submitted the job, cluster where the job will run, current status of the job, timeout, requested number of processors, and other parameters – some of which (e.g., job status) will change dynamically throughout the job's lifetime, as described in subsequent sections. `Submit_Click` then proceeds to create a job directory, named after the job ID, within the network file system accessible from all the compute nodes of the cluster (e.g., `H:\CBSU\JOBS\12345`, where job ID is `12345`), and saves the input files in this directory. Before the job can be submitted, `Submit_Click` has to select a proper cluster using the BioHPC built-in meta-scheduler, or, if a specific cluster was requested by the user, it has to verify that this cluster is appropriate for the job (for example, that the requested number of processors does not exceed the total number of processor in the cluster). The procedure then constructs a DOS batch script, usually called `run.bat`, to be submitted via cluster scheduler and eventually to be run on one of the compute nodes the scheduler will assign to the job. The script consists of commands that first create job-specific temporary directories on local scratch disks on all compute nodes assigned to the job, then copy all the necessary input files from the job

directory to the local scratch directories, launch the actual application executable, and then copy any output files from the local disks back to the job directory. Section 8.1 outlines general structure of this script. Finally, after `run.bat` is constructed and saved to the job directory, `Submit_Click` calls another procedure, which interfaces to the cluster scheduler and actually places `run.bat` in the cluster queue on behalf of the **Interface Domain User (IDU)** – a special account defined in Windows domain. The job status field in the database is set to **QUEUED** and a notification e-mail is sent to the user. At this point, the user is presented with an info page reporting on the submission and containing links to the output and log files. The same links appear in the e-mail notification message. With the help of these links, the user will be able to monitor progress of the job and retrieve output files when the job ends (more on file retrieval and job monitoring - in the Sections 2.2.3 and 2.2.5, respectively). The work of the submission page and its "code behind" is now complete and the job is handed over to the cluster scheduler.

It should be stressed that any distinction between the users is effective only at the level of BioHPC. All jobs, regardless of which users created them in BioHPC, are seen by the cluster scheduler as belonging to a single Windows domain account, the IDU. This domain user also owns all the processes running on the web server and compute clusters on behalf of BioHPC (web pages, auxiliary programs, applications, etc.). The information about the original owner/creator of the job is stored in the `jobstatus` table in the `CBSUJOBS` database and utilized only by BioHPC. This way, the cluster resources can be made publicly available without setting up separate domain accounts for all users.

2.2.2 Job identification and data privacy

The most important entries in `jobstatus` table are the **job ID** and the so-called **control number**. The job ID is simply a unique integer incremented by 1 each time a new job is created. The control number is a large signed random integer generated and recorded in `jobstatus` at the time of job creation. Although job ID alone would be sufficient to identify a job, all operations on job data (files or database entries) within BioHPC are performed by web programs (`*.aspx`) which require **both** the job ID and the control number to be supplied as arguments. Prior to performing any actions, these programs verify that the control number they get as an argument does match the one originally recorded in the `jobstatus` for a given job ID. While job ID is essentially public information, the matching control number is known only to the user who submitted the job (and to BioHPC administrators and programs run by them). Thus, the control number can be thought of as a password to the resources of the job with a given job ID. All this does not mean that the user is forced to remember bulky control numbers for all his jobs. These numbers are hidden quite deeply, embedded in the HTML links the user gets in his notification emails. Simply clicking on such a link ensures that both job ID and the matching control number are properly passed on to the appropriate web program. Subsequent sections give examples of this policy at work.

2.2.3 File access and job control

Once the job, or more specifically – the `run.bat` script is placed in the cluster queue by the `Submit_Click` procedure, it starts living its own life as determined by the cluster scheduler. Typically, it will wait a while for the nodes to become available, then it will start, and finally it will end or get killed by the cluster scheduler if the timeout is exceeded. While all this happens independently of BioHPC, the users and administrators need to be able to use BioHPC to check the status of the job (is it

still waiting, running, or did it finish?) and to interact with the job both while it is still running and after it completes. Interactions of a user with a job, typically involving file access and canceling, are described in this Section.

Each job produces application-specific output files stored, upon job completion, in a job-specific **job directory** on a network file system. From the point of view of the operating system, all these files belong to the IDU domain account. Access to these files is provided to the original job owner using a web page `showfile.aspx`, launched from the user's browser and running on the server on behalf of the IDU. A typical invocation of the `showfile.aspx` program is of the form of an http link similar to

```
http://BioHPC_location/showfile.aspx?jobid=25240&cntrl=532239145&fileid=2&mode=show
```

Links like the one above are embedded in the e-mail notification messages sent to the users at various stages of the job's "life" in BioHPC, so that all that's required to launch `showfile.aspx` is a click of a mouse. The integer parameter `fileid` is translated by `showfile.aspx` into the name of the file to retrieve. The translation is done by calling the application-specific public method `FileName` – a part of the "code-behind" of the application's submission page. A notification e-mail can contain many different links to `showfile.aspx` pointing to different files specific to a given application. The file indicated by argument `fileid` is presented to the user in the way specified by the argument `mode`. In this example, `mode=show` will display the file in a browser window. Other modes supported by `showfile.aspx` allow to download the file from the job directory using the http protocol (`mode=http`), or from the job directory replica on the ftp server, if present, using the ftp protocol (`mode=ftp`). It can be seen that both job ID and the control number are among the parameters passed on to the program. In the beginning, `showfile.aspx` verifies that the control number given as an argument matches that recorded in the `jobstatus` table for this particular job id. Access to the file is granted only after successful control number verification. Since the notification e-mails about a given job are only sent to the owner of this job, he/she is the only one (besides administrators) in possession of the control number, which ensures data privacy.

Another web interface program, called `cbsujob.aspx`, allows the users and administrators to control some aspects of a job after it is submitted or after it completes. A call similar to

```
http://BioHPC_location/cbsujob.aspx?jobid=25240&cntrl=532239145&fileid=2&action=cancel
```

can be used to control (in this case: cancel) the job with a given job ID and the matching control number. The values of the parameter `action` include

- `cancel` – cancel the job
- `restart` – requeue the job
- `stop` – stop the job without changing its database status
- `changestat` – change status of the job in the database
- `archive` – remove from the list of jobs shown by monitoring tools – see Section 4.2.
- `userrestart` – requeue by a user
- `imstop` – manipulate the file IMrun – specific for the application program IM

- `showstats` – show the database entries of the job: current status of the job, cluster it is/was queued on, timeout, start/stop date, etc.
- `datadel` – delete all the job's data from the BioHPC fileserver and ftp server, if present.

Most of the `actions` are general and independent of the application. Others (like `restart`) have to be customized for some applications and so the code of `cbsujob.aspx` has to contain some application specific routines. While `cancel`, `showstats`, `datadel`, and `imstop` are available to all users, the other actions are generally reserved for administrators only and executed from within the administrative web pages. As it was the case with the `showfile.aspx` program, the appropriate `cbsujob.aspx` links are embedded within notification e-mails and administration pages.

2.2.4 Keeping track of job status

An important aspect of BioHPC is to provide users with up to date information about the status of their jobs. As mentioned before, users and administrators can retrieve this information from the `jobstatus` table of the `CBSUJOBS` database. To ensure that this **database job status** is accurate, it has to be reconciled with the **scheduler job status**, as obtained by querying the cluster scheduler. Here we focus on how BioHPC keeps the database and scheduler job statuses coordinated and how users are notified about status changes.

As mentioned before, the `Submit_Click` function of the submission page leaves the job marked as **QUEUED** in the `jobstatus` table. To keep the job information up to date, the database status needs to be changed once the job starts executing. This change is accomplished by a program called `cbsufinalize.exe`, executed by the `run.bat` script shortly after the latter starts, but before it launches the actual application executable. The syntax of the command is

```
cbsufinalize.exe <jobID> <0 | 1> <timeout> <cntr_num> [nomail | send | sys]
```

where the last (optional) argument controls if and to whom an e-mail notification will be sent. The parameters passed on to `cbsufinalize.exe` include the indicator 0 or 1, depending on whether it is called in the beginning or at the end of `run.bat`, respectively, the job ID and control number (both generated earlier within `Submit_Click`), and also the timeout in minutes (see Applications administration for more info about timeouts). The `cbsufinalize.exe` program contacts, in turn, the web program `cbsufinalize.aspx` running on the web server. The latter program recognizes the parameter "0" as indication that the calling `cbsufinalize.exe` marks the beginning of `run.bat` and thus it updates the database status of the job (as defined by job ID and the control number) to **RUNNING**. If the last parameter in the call to `cbsufinalize.exe` is omitted (as it is in this case), or set to `send`, an e-mail is sent to the user notifying him about the start of the job.

The advantage of having a web application (`cbsufinalize.aspx`) do the task of updating the database is that it can be easily called even from remote locations, such as JSDL-connected clusters. Despite this openness, the chance of compromising the database by unauthorized user is rather small. This is because before contacting the database, the program verifies that the control number passed on

as a parameter matches the control number generated for this job and recorded in the database at the time of submission. This is an example of job identification policy (described earlier) at work.

The `cbsufinalize.exe` program (and `cbsufinalize.aspx`) is called once again, this time at the very end of `run.bat`, after the application program finishes running. The indicator flag is now set to "1", which tells `cbsufinalize.axpx` to change the database status of the job from **RUNNING** to a finished state. There is, however, not just one state a job may end up in when it finishes running. The `cbsufinalize.aspx` program is able to distinguish between several such end states:

- **FINISHED CORRECTLY**: when everything went fine and all output files are present and in order.
- **FAILED/USER INPUT**: when the job failed due to some user-generated errors in input data.
- **FAILED**: when the job fails for unknown reasons. In this case, an e-mail notification will be sent to the administrator rather than the user.
- **FAILED/stalled**: when the job failed for unknown reasons and output files have not been updated for a long time. An e-mail notification will be sent to the administrator rather than the user.
- **FINISHED PREMATURELY**: when the job did not complete, but generated output files and there was no obvious error message generated (this situation is typical in case of a timeout).
- **FINISHED PREMATURELY/user notifed**: when an e-mail notification was sent to the user after job FINISHED PREMATURELY; when run from `run.bat`, `cbsufinalize.aspx` will send such message out automatically.

The decision about which of these states to put the job in is made by `cbsufinalize.aspx` based on the analysis of output files generated by the job. Since different applications generate different kinds of files, such an analysis is, in general, application-specific. It is accomplished by calling an application-specific procedure `Jobstatus`, which is a part of the "code-behind" of the application web page in question, and is defined as a "public static" method (i.e., available from other web pages, including `cbsufinalize.aspx`). `Jobstatus` scans the job directory searching for output files which a given application should have generated, analyzes these files searching for error messages which may indicate that the user made a mistake in the input, and returns an indicator which allows `cbsufinalize.aspx` to update the job's database status with the proper flag and send the appropriate e-mail message. This message is usually sent to the user who submitted the job, except for the case when the end status is determined as **FAILED or FAILED/stalled**. In these cases the e-mail is sent to the BioHPC administrators urging them to look closer at the reasons of the job's failure. These reasons may include hardware failures, problems with the application executables, or an input error condition previously unaccounted for by the application's `Jobstatus` procedure. After troubleshooting the cause of the failure the administrator has an option to change the job's status (for example, to **FINISHED** or **FAILED/USER INPUT**) and notifying the user, or to re-queue the job (for example, after a hardware problem is fixed). See Job administration for more information about job control.

Special attention must be given to the case when a job times out and is killed by the scheduler before the final call to `cbsufinalize.exe` can be executed. The database status of such job is still

RUNNING, although it is no longer in the cluster queue. BioHPC handles such situations with the help of an auxiliary program called `stardaemon.exe`. This program, run periodically as a Windows "scheduled task" on the web server, scans the `jobstatus` table for jobs with database status **RUNNING**, but which cannot be found in the cluster queue. For each such job, `stardaemon.exe` executes the corresponding `cbsufinalize.exe` command, i.e., it finishes off the work that could not have been finished by `run.bat` due to timeout. Another function of `stardaemon.exe` is to automatically archive (see Job monitoring and administration/**Active?**) finished jobs older than a number of days specified when running the site setup program `setup.aspx` (see **INSTALLATION**).

Finally, database job status updates can also result from actions (like `cancel`, `restart`, or `changestat`) taken by the job control interface `cbsujob.aspx`, described earlier. In such cases, the latter program handles those updates.

2.2.5 Job monitoring

As mentioned before, the primary way BioHPC communicates with a user is through e-mail notifications sent each time a job is submitted, when it starts executing, and when it ends. The links to `showfile.aspx` and `cbsujob.aspx` (with appropriate parameters) allow for checking of the job status and provide access to the log and output files, which can be used to monitor job progress. A link is also provided which allows a user to cancel the job.

The job monitoring task is even simpler for **registered users** (for details - see User Administration), who have access to a handy web tool **MyJobs**. This tool collects and presents the information about all the jobs (present and past) of a given user, displaying the current job status and links to output files. **MyJobs** essentially eliminates the need to handle multiple e-mail notifications, which may become a problem if BioHPC resources are used frequently. An extended version of this tool, available to BioHPC administrators, is described in Job administration.

2.2.6 Caching cluster scheduler information

Several components of BioHPC call cluster scheduler APIs in order to retrieve information about queued and running jobs and the number of nodes currently occupied and available on each cluster. This information is needed, for example, upon job submission, when BioHPC is trying to decide which cluster to submit the job to. It is also used by job-monitoring tools, such as **MyJobs**, which display the job status. Retrieval of information from cluster schedulers may be a time consuming task, especially if several big and busy clusters are involved, and may significantly slow down the BioHPC functions mentioned above. To remedy this slowdown, BioHPC uses a caching mechanism. The information obtained by calling scheduler APIs is stored in a SQL table (called `cache`) in the `CBSUJOBS` database for quick retrieval and typically this is where the requesting BioHPC functions take it from. However, by checking the timestamps on the cache records, these functions can determine when the cached information is too old. In such a case, they contact the schedulers directly through the APIs and then update the selected cache records for future use. This happens also if the requested information cannot be found in the cache. In this way, the cache is constantly updated by the functions that use it. A cache record will be updated if it is older than the cache timeout parameter (**cachetimeout**) specified when running the site setup program `setup.aspx` (see **INSTALLATION**). Our recommended value of this

parameter is 15 minutes. Increasing this timeout will make BioHPC run somewhat smoother, but the job listings and submission decisions may be based on cluster information which is not completely up to date. Setting **cachetimeout** to zero will cause BioHPC to bypass the caching mechanism and always obtain current job/node information directly from cluster schedulers.

In addition to the dynamic update of the cache by the requesting functions, BioHPC uses a special utility program called `cache.exe`, which (similarly to `stardaemon.exe`) is set to run periodically on the web server as a scheduled task. Each time it runs, this program refreshes the `cache` table completely. Since `cache.exe` runs in the background, independently of the BioHPC web pages, it does not slow these pages down.

2.2.7 Displaying application statistics

A link Application Statistic in the (MISCELLANEOUS category) displays a table summarizing the numbers of job submissions and failed jobs for all applications.

3 Installation

This Section is still under construction. A comprehensive sketch of the installation procedure can be found at http://biohpc.com/inst.aspx.

As a part of the installation process, a web program `setup.aspx` has to be run to customize the BioHPC site by defining certain vital parameters in the `web.config` file. Due to confidential nature of some of these parameters, `setup.aspx` will only open on the web server where the site resides. Here is the summary of the parameters the program will ask for:

General Settings

Address: URL of the site, e.g., `http://my_biohpc_site/`

AddressTestSite: URL of the developer's version of the site, e.g., `http://my_biohps_site/tst/`

TestSite: `TRUE` (if this `web.config` is a part of developer's version) or `FALSE` (otherwise)

Title: title string to be displayed on top of each of BioHPC pages (e.g., `My BioHPC site`)

Subtitle: subtitle string to be displayed underneath the title (e.g., `my favorite site`)

Logo: a picture to be displayed in the top-left corner of each of BioHPC pages (e.g., `mylogo.jpg`)

BioHPCMail: contact e-mail address for users to communicate with the site admins

BioHPCAdminMail: e-mail address(es) of the site administrators, where various BioHPC-generated messages will be directed (such as those about failed jobs)

SMTPServer: name of the machine responsible for handling e-mail

FtpJobsUrl: URL of the job directory on ftp server, if present, e.g., `ftp://my_ftp_srv.my_domain.edu/JOBS/` (optional)

FtpPblastUrl: URL of the PBLAST directory in the ftp server, if present, e.g., `ftp://my_ftp_srv.my_domain.edu/PBLAST/` (optional); this directory is typically used by selected users to upload large files, not suitable for http transfer

FtpJobsUnc: UNC of the job directory on ftp server, if present, e.g., `\\my_ftp_srv\share\JOBS\` (optional)

FtpPblastUnc: UNC of the PBLAST directory on ftp server, if present, e.g., `\\my_ftp_srv\share\PBLAST\` (optional)

Organization: name of your organization, e.g., `CBSU`; will be displayed on application web pages (in the sentence informing the user where his calculations will be carried out).

OrganizationURL: URL of your organization's web page; will be displayed on application web pages (in the sentence informing the user where his calculations will be carried out).

WebDir: directory on web server where BioHPC is installed, e.g., `T:\Inetpub\wwwroot\BioHPC\`

InterfaceUserName: or Interface Domain User (IDU) – a domain account to own all files, processes, and jobs generated by BioHPC, e.g., `my_domain\biohpc_user`

InterfaceUserPass: password for IDU

Zip: path to the `zip.exe` program on the web server, e.g., `H:\CBSU\unixutils\zip.exe`. Needed by programs serving remote clusters accessed through JSDL and ftp.

UnZip: path to the `unzip.exe` program on the web server, e.g., `H:\CBSU\unixutils\unzip.exe`. Needed by programs serving clusters connected through JSDL.

jt.exe: path to the `jt.exe` program on the web server, e.g., `H:\CBSU\jt.exe`, this program (downloadable from Microsoft) is a part of the real-time (RT) scheduler (described in RT scheduler – under construction)

perl: path to perl executable on the web server, e.g., `C:\Perl\bin\perl.exe`

vsched: path to Velocity scheduler executable, e.g., `C:\Program Files (x86)\Velocity\` (optional, needed only if clusters running Velocity are present)

RT_timeout: timeout (in minutes) for real-time (RT) jobs

autoarchive: number of days; jobs older than this and having unambiguous end status will be automatically archived (see Job monitoring and administration/**Active?**) by the `stardaemon.exe` scheduled task (see Keeping track of job status).

cachetimeout: e.g., `900`; given in seconds; when a list of queued and running jobs and/or node status information is obtained by contacting the cluster scheduler, it is stored in an database cache from which it can be quickly retrieved (without contacting the scheduler) within **cachetimeout** minutes of the original call. If the current cluster status information is needed after **cachetimeout**, the cluster scheduler will be contacted again to refresh the cache. If **cachetimeout** is set to zero, the cashing mechanism will be bypassed and job/node information will be retrieved by calling scheduler APIs directly each time.

DBdirPrivate: directory to store user's private BLAST databases, e.g., `H:\CBSU\blastdata2`

echeck: whether to enable e-mail verification (`ON` or `OFF`)

echeck.timeout: timeout (in days) for cached e-mail addresses

permissive_https: parameter used for remote clusters connected through JSDL.

stat_cont_date: default: blank. Used to display job submission statistics (page `cbsujobstat.aspx`). If left blank, only jobs submitted through this installation of BioHPC will be counted in the statistics. If set to a date, e.g., 6/13/2003 (prior to inception date of this BioHPC installation), job statistics will include the pre-BioHPC period starting on stat_cont_date. To use this option, the pre-BioHPC data (number of jobs and number of failed jobs) have to be entered in columns `statcorr` and `statcorr_f`, respectively, of the table `apps` in `CBSUJOBS` database.

servertemp: e.g. T:\ - directory on the web server, where some application submission programs will create their temporary files and subdirectories, needed, for example, for input verification.

Miscellaneous links

optional; these links will show up at the bottom of the right-hand-side bar on each BioHPC web page, underneath the application links. The format is <description>|<url>, for example, `CBSU Home|http://cbsu.tc.cornell.edu/`

SQL Database connection for BioHPC

server hostname: name of the machine running MS SQL Server, e.g., `my_sql_srv`

database name: for example, `CBSUJOBS`

user id: owner of the database as configured in MS SQL Server, e.g., `cbsujobs_owner`

password: password for database access

workstation id: name of the web server, as defined in the Windows domain.

persist security info: e.g., `True`, SQL server connection parameter

packet size: e.g., `4096`, SQL server connection parameter

The program `setup.aspx` can also configure settings for sequence data retrieval from the MS SQL database (if application `RetrieveSeq` is installed which uses such sequence database) as well as the parameters of clusters connected through JSDL protocol (if present).

4 BioHPC administration

The administrative pages can be accessed from any BioHPC page by clicking on the link **Administration** on the top bar. This link is only available for registered users with administrator status. A default administrative user, called `admin@biohpc` (password: BioHPC4All), is defined during BioHPC setup – it can and should be changed immediately after the setup is complete (see User administration for details).

What opens up is after clicking on the **Administration** link is a page containing links to all administrative functions of BioHPC. In the following, these functions will be described in detail.

4.1 User administration

Information about the registered users is held in the table `users` in the `CBSUJOBS` database. Tables `useraccess` and `userdbaccess` determine access different users have to different applications and databases, respectively. A related table is `echeck`, which holds information about the recently

used e-mail addresses (*vide infra*). The user administration pages access and modify these tables. For detailed description of the tables please consult the Appendix.

Clicking on **Manage users** button opens a list of all users registered with BioHPC, with basic information about them. The display can be adjusted by applying filters over any column and/or checking/un-checking individual users. The user can be deleted using the corresponding **Delete** link the user's information can be edited using the **Edit** link. The editable fields are: e-mail address (this will serve as the user's "login name" and as such is required), full name, institution, group (all optional – introduced for organization purposes), and status. The latter field determines whether the user is a regular registered user (Active), an administrator (Admin), or a registered user whose access has been blocked (Inactive). The Edit link can also be used by an administrator to change the user's password. The user ID cannot be edited – it is assigned automatically by the database upon user creation. After filling up/adjusting the editable fields – click **Submit Changes** button to update the `users` table.

After updating basic user information under the **Edit** link, it is possible to configure the user's application and database access, using the **Configure Apps** and **Configure DB** buttons, respectively. For example, **Configure Apps** will open a list of all available applications, each with a checkbox next to it. Checking the checkbox will make the corresponding application available to the user (after **Submit Changes** is clicked, which will produce an entry in the `useraccess` table). Applications declared as "available to everyone" (see Application administration) will have their checkboxes automatically checked and shaded out (not modifiable). **Configure DB** button works in an analogous way, with databases instead of applications and table `userdbaccess` instead of `useraccess`.

The main **Manage users** page features a set of action buttons located above the user list:

The button **Add New User** will open a dialog with basic information fields to enter for a new user, the most important one being the e-mail address. A randomly generated initial password will be provided (which can be edited, if desired). Clicking **Add User** will generate a new entry in the `user` table (i.e., create a new user), providing a unique numeric user ID, and send a short "welcome e-mail" to the new user containing the initial password and instructions on logging in and changing the password. The application and database configuration buttons will also be displayed, allowing the proper adjustments to be made for the new user.

The buttons **Edit Checked**, **Apps Checked**, and **DB Checked** allow to modify the affiliation information and status, applications, and database access, respectively, of the whole group of users (the "checked" ones). Multiple users can be added at the same time (via **Add Multiple Users**), deleted at the same time (via **Delete Checked**), or e-mailed at the same time (**Email Checked**). Using the **Post Message** button an administrator can post a general message visible on all BioHPC pages (for example, about the upcoming cluster maintenance), located on top of the page. The message should be entered in HTML in the text box provided. An existing message can be removed using the **Clear** button.

The first two entries in the users table are reserved for two special users: "guest" and "deleted" (these users will **not** be displayed in the user table in **Manage Users**). The user "guest" (with user ID equal to 1) represents all users who are not registered. If somebody accesses BioHPC and does not log in (because

he is not registered or because he does not want to log in), his session is automatically treated as opened by the "guest" user. The user "deleted" (user ID equal to 2) has been defined in order to inherit all job entries orphaned by users who submitted the jobs as registered users, but have since been removed from the registered user database.

An important aspect of user administration is verification and **handling of e-mail addresses**. A valid e-mail address is essential for the communication between BioHPC and users. Therefore, an e-mail address provided by the user at submission time is first verified (from within the submission page) by contacting the user's e-mail server. If errors occur, the job is not submitted. If, on the other hand, verification is successful, the address is stored in table `echeck`, which plays a role of a cache of valid e-mail addresses. Next time this address is requested, it will be immediately accepted based on its presence in cache rather than by verifying the e-mail server again. Since the e-mail verification process is not perfect, some e-mail addresses need to be recognized as valid even if they fail the verification. Such addresses can be manually entered into the table `echeck` and marked as "trusted". A web tool for maintaining the e-mail address cache and the list of trusted e-mails can be accessed from the main administration page through the **E-mail check** button. The tool also provides a way to verify e-mails independently of job submission pages.

4.2 Job monitoring and administration

Monitoring and controlling the flow of jobs submitted to the clusters is the most frequently used administrative feature of BioHPC. It is accessible to any administrative user directly from any BioHPC web page using the link **Manage jobs** on the upper bar, as well as through the main **Administration** link. The link takes us to the **Job Admin** page containing a table where rows correspond to the active (this term will be defined later) jobs and columns represent most of the fields found in the database table `jobstatus`. Thus, the table contains full specification of each job, including the current status. What's even more important, it also provides job control functions (such as manual status change, cancel, and requeue) as well as direct access to job directories. To focus attention, jobs being displayed can be filtered according to any subset of parameter fields. Operations on multiple jobs (those "checked" using the checkboxes in the leftmost column of the table) may be performed with just a few mouse-clicks. At the bottom part of the page, the queued and running job information is summarized on a per cluster basis. This allows for a quick "at-a-glance" assessment of the current load and job distribution among various clusters. The **Job Admin** page will refresh automatically every 15 minutes. It is a common practice among the BioHPC administrators to have this page always open to stay on top of things.

The meaning of various columns in the **Job Admin** table is as follows:

ID: the job ID automatically generated at the time of submission. It uniquely identifies the job to BioHPC and does not change even if the job is re-queued.

Appname: the name of the application, as configured in Application administration.

Submitted: time when the job was first submitted. Will not change even if the job is later re-queued.

User: e-mail address of the user who submitted the job. Clicking on the address will open an e-mail application ready to send a message to this user.

Cluster: cluster where the job is (or last was) in the queue. May change if the job is re-queued on another cluster. The cluster originally selected by the user at submission time is given in parentheses (this may be "Auto" if the user relied on BioHPC to select the cluster).

Nodes/(CPU): in parentheses – the total number of processors (cores) the job is occupying on a given cluster. Some MPI applications may be configured (in Application administration) to exclusively occupy whole multi-core nodes – in such a case the number of such nodes is shown outside of parentheses and the symbol "x" is displayed to indicate the exclusive character of the job.

Cluster job ID: an identifier assigned to the job by the cluster scheduler (different from the unique BioHPC job **ID**); will change when the job is re-queued.

Timeout: time (in minutes) after which the job will be killed by the cluster scheduler even if not yet finished.

Started: time when the job started executing. This field is empty for jobs with status **QUEUED**.

Ending: for finished jobs – the time the job finished or the time of the last status change (which may have been manually done by an admin), whichever is later; for running jobs – the latest anticipated end time (i.e., **Started** + **Timeout**). This filed is empty for jobs with status **QUEUED**.

Jobname: name of the job, as specified by the user at submission.

Status: current status of the job, determined from the entry on the `jobstatus` table and from the job listing obtained from the cluster scheduler (typically, these are cached – see Section 2.2.6). Some of the possible values of job status have been discussed in Section 2.2.4. The complete list is as follows:

- **SUBMITTING**: the job is listed as "queued" in the CBSUJOBS database, but not yet known to cluster scheduler.
- **QUEUED/SUBMITTED**: the job is listed as "queued" both in the CBSUJOBS database and by the cluster scheduler.
- **RUNNING**: the job is listed as "running" both in the CBSUJOBS database and by the cluster scheduler. If the output files generated by the job did not change recently (as specified in the **Logfile timeout** filed in Application administration), the label "stalled" will also be displayed. This is to bring it to the administrator's attention that something may be wrong with the job.
- **CANCELED**: the job has been canceled by user or administrator; it is no longer in cluster queue, and its database status is "canceled".
- **FINISHED CORRECTLY**: when everything went fine and all output files are present and in order.
- **FAILED/USER INPUT**: when the job failed due to some user-generated errors in input data.
- **FAILED**: when the job fails for unknown reasons. In this case, an e-mail notification will be sent to the administrator rather than the user.
- **FAILED/stalled**: when the job failed for unknown reasons and output files have not been updated for a long time. An e-mail notification will be sent to the administrator rather than the user.

- **FINISHED PREMATURELY**: when the job did not finish, but generated output files and there was no obvious error message generated (this situation is typical in case of a timeout).
- **FINISHED PREMATURELY/user notifed**: when an e-mail notification was sent to the user after job FINISHED PREMATURELY.
- **MAINTENANCE**: when a cluster is put in a maintenance mode (see Cluster administration), the jobs marked in the database as running on that cluster will show up as "MAINTENANCE *", the star indicating that the database status of the job is still "running". The database state can be changed (see description of **Display and update** button later on) to "MAINTENANCE" proper to mark this job for further consideration (e.g., to be re-queued later). If this change is not done and the cluster comes back up with the jobs still reported as running by cluster scheduler, they will show up as RUNNING again.
- **UNKNOWN/update pending: used for transition status with JSDL clusters**

Occasionally, any of the "finished" states can be shown augmented with a star ("*"). This may happen when the database status of the job is "running", but the job is no longer reported by the cluster scheduler as being in the queue. In such a case, the **Job admin** tool is guessing the final state of the job by analyzing the output files with the help of the `Jobstatus` function, specific for a given application, in a manner analogous to how the `cbsufinalize.aspx` utility works. The star denotes this guessing process. Normally, the database status is updated to this guessed status by the `stardaemon.exe` program running periodically as a scheduled task (see Keeping track of job status), so that it is only on rare occasions that the stars can be seen in **Job Admin**. An exception is the "MAINTENANCE *" status, which will not be updated to MAINTENANCE" by `stardaemon.exe`. For this purpose, one has to use the **Display and update button** (discussed later in this Section).

Active?: an "active" is a job which has not been put in an "archived" state. Technically, "archiving" a job is equivalent to adding 1000 to the numeric representation of the job's status in the `jobstatus` table. Unless specified here, the **Job Admin** page displays only jobs which are not "archived". An administrator can "archive" jobs which are finished, old, and no-longer of interest for the sole purpose of avoiding clutter in the **Job Admin** table. Jobs older than certain number of days (see description of `setup.aspx` in **INSTALLATION**) are automatically archived by the `stardaemon.exe` utility program. "Archiving" a job does not affect its jobs directory or files in any way. "Archived" jobs information can still be displayed by **Job Admin** page if the filter in this field is set to **No** or **Any**.

Files: collection of links to various job-related files. The link log file: show typically displays the standard output from the application and therefore is best suited for monitoring the progress of the job. The link output will display the notification page as sent to the user by e-mail. The link DIR gives direct access to the job directory via Windows file explorer (only available on machines within the Windows domain).

Action: collection of links providing job control features.

- STOP: (for running jobs only) will *cancel* the job (i.e., remove it from the cluster queue and update the status to "CANCELED" in the `jobstatus` table), or *stop* the job (remove it from the cluster queue without touching `jobstatus`).

- ARCHIVE: (for jobs NOT in the RUNNING state) will "archive" the job (see description of the field Active? above).
- DEL: (for jobs NOT in the RUNNING state) will open a dialog allowing all the files of this job to be deleted from both the file server and the ftp server.
- RESTART: will re-queue the job; one can select a new cluster (from among the clusters available for a given application), timeout, the number of processes (for MPI applications), whether the job needs to run alone (exclusively) on all nodes it uses (0 for non-exclusive, 1 for exclusive), and whether to send a notification e-mail to the user.
- STAT: manually change database status of the job to **FINISHED CORRECTLY** (SC), **FAILED** (FL), **FAILED/USER INPUT** (FU), or **FINISHED PREMATURELY** (PM). Note that clicking PM will not notify the user of the status change – the proper NOTIFY option must be used for this purpose.
- NOTIFY: notify the user by e-mail about queuing of the job (S), start of the job (0), correct end of the job (1), failure of the job (E), or premature end of the job (P).

Functions available through action buttons on the **Job Admin** page:

Display: force page refresh. Normally, the page will refresh itself every 5 minutes, unless the corresponding control checkbox is un-checked.

Clear Cache & Display: clear the cache of job and node information collected from cluster schedulers and display refreshed data; this automatically leads to re-population of the cache.

Display & update: archive all jobs with unambiguous end status (to un-clutter the display table), update the database status of all finished (i.e., not in cluster queues) jobs which have not yet been handled by `stardaemon.exe`, including sending e-mail notifications to users whose jobs ended prematurely, and refresh the page.

Refresh checkbox: if checked, the page will be automatically refreshed every 15 minutes.

Unlock Admin: remove any locks set up by various BioHPC administrative functions during job status update operations; typically, this button needs to be used only if such update operations failed for some reason.

Status change checked: manually change database status of all "checked" jobs.

Stop/Cancel checked: stop (remove from cluster queue without changing database status) or cancel (remove from cluster queue and change database status to "canceled") all "checked" jobs.

Restart checked: re-queue all "checked" jobs. A dialog will be presented where the admin can specify the following actions/parameters:

- whether or not to send and e-mail notification to the user when job starts; such e-mail may not be necessary, for example, if the job being re-queued crashed due to hardware problems on the cluster
- for each job, use the same submission parameters (cluster, number of CPUs, and timeout) as when the job was initially submitted;

- use different submission parameters; in this case only the “checked” parameters will be changed (to the same value for all jobs being re-queued), the un-checked ones will remain the same (and still job-specific) as when the jobs were first submitted.

Most frequent use of the **Restart checked** button is to move a bunch of jobs queued on one of the clusters to another cluster with free nodes.

Fix checked: gives direct access to the database entries for the selected jobs exactly as recorded in the `jobstatus` table. All entries can be manually edited. Detailed knowledge of the structure of `jobstatus` table (to be described in the Appendix) is required to use this option. Direct access to the `jobstatus` entries for selected jobs is also provided directly from the **Administration** page, via the **Fix jobs** button.

Delete checked: removes all the files related to the checked jobs from both the fileserver and the ftp server.

An important aspect of job management is the **cleanup of old job data**. The output files generated by jobs submitted through BioHPC are stored in the job directories on the file server, and (optionally) are also replicated on the ftp server. Once in a while, directories corresponding to old jobs will need to be removed to make space for the new jobs. The web tool for old job data removal, `cleanJOBS.aspx`, is available from the **Administration** page by clicking on **Remove old jobs data button**. Here, the admin can specify the time threshold and click on **Remove data files** to clean up space occupied by jobs older than that threshold. What actually happens is that a special job is submitted to one of the clusters (selected from a dropdown list) launching a program `clean_old.exe`, which performs the actual clean-up work. Because of this job-like character of the cleanup operation, the `cleanJOBS.aspx` page is treated like submission page of an application. It is configured as “hidden” (which makes it not available through a public link) and accessible only to registered users (see Application administration for more info on how to configure applications). Additionally, the “code-behind” allows the page to be executed only by users with administrative privileges.

It should be stressed that removing old jobs data from the file server of ftp server does not remove the corresponding records from the `jobstatus` table in the `CBSUJOBS` database. These records will stay there indefinitely.

A similar problem sometimes occurs on remote clusters (see Section 6). Normally, the remote job directory which stores files belonging to this job for its duration is removed after the output data is transfer back to the main BioHPC job directory. However, the removal process may sometimes fail, leaving the unused remote job directory behind. Over time, such stray directories may accumulate on a remote cluster and take up a significant amount of disk space. To address this issue, BioHPC provides a clean-up tool available by clicking the **Remove old JSDL data** button from the main administration page. Using this tool, an administrator can list the expired remote job directories on all remote clusters and remove them. Unlike in the case of the old data removal from the BioHPC file server, the cleanup of the remote clusters is performed directly via ftp rather than by submitting a special clean-up job. Since the

number of left-over remote directories is usually fairly small, the clean-up process is short and a simple ftp session is sufficient.

4.3 Cluster administration

Clicking in the link **Administration** and then on the button **Manage clusters** will produce a list of clusters currently defined within the BioHPC installation.

The **Edit** links in the rightmost column of the table allow an administrator to configure settings for each cluster. These settings, stored in table `clusters` in the `CBSUJOBS` database, are the following:

- **Database ID**: unique integer, automatically generated when the cluster is configured.
- **Name**: name of the cluster
- **Status**: a choice of
  - **Full maintenance**: the cluster is off-line; if this is set, a window will appear where a message to users can be entered; this message will be visible in the message box at the bottom of each BioHPC web page; change of status to "Operational" will erase the message.
  - **Partial maintenance**: left for future development
  - **Operational**: cluster is working normally and accepting jobs
- **Scheduler**: the following schedulers are currently supported
  - vsched: custom Windows scheduler developed at Cornell Theory Center
  - microsoft: native scheduler of Compute Cluster Server (CCS)
  - JSDL: for remote clusters connected through JSDL (see Section 6).
  - RT: native scheduler of BioHPC serving real-time clusters (if present). More info about the RT clusters – in Section 5.
- **MPI command**: the beginning of the command used to launch MPI applications. On clusters where the `H:\` drive is permanently mounted this should be set to `mpiexec -machinefile machines.nodes`. Otherwise (e.g. on most JSDL-connected clusters) the proper command would be `mpiexec -env CBSUH %CBSUH% -env CBSUT %CBSUT% -machinefile machines.nodes`. On clusters running **vsched** scheduler with **MPIPro** the command is simply `mpirun`. Vsched clusters have to have the `H:\` drive mounted on each node. Other options (such as `-np` or `-wdir`) will be appended to this string when `run.bat` is constructed.
- **Binaries path**: directory where sequential binaries are stored; must be visible from all nodes of this cluster.
- **Ptools path**: directory where the parallel executables (launched through MPI) are stored; must be visible from all nodes of this cluster.
- **Extra string**: reserved for future development
- **CBSUH**: root of the shared network drive where the job directories will be stored. Can be specified as UNC name. If left empty, `H:` will be assumed, in which case the network drive will have to be mounted as `H:\` on each node of the cluster.
- **CBSUT**: local scratch drive on each of the compute nodes. DOS variables (such as `%TEMP%`) accepted. If left blank, `T:` will be assumed.
- **Maxnodes**: maximum number of nodes that can be utilized by BioHPC on this cluster; if zero – no limit will be assumed.

- **CPUs per node**: number of CPU cores on each node of this cluster. This parameter is used mainly to determine the correct form of scheduler submission command. The number of instances of a parallel application that can be run on each node is application-specific and configured in **Manage applications**.
- **RAM per node (GB)**: memory available on the node – this entry is currently not used
- **HD per node (GB)**: size of the local scratch disk space on the node – this entry is currently not used
- **Priority**: integer number, the priority in which this cluster will be considered by the internal BioHPC metascheduler; the lower the number the higher the priority.

The **Edit ACL** links in the rightmost column of the table allow an administrator to configure access control list (ACL) for each cluster. The ACL determines which users can submit jobs to the given cluster. The options are

- **Unrestricted access**: all users are allowed
- **Registered users only**: all registered users will be allowed access
- **Only users listed below**: access will be restricted to a specific subset of registered users identified by their user IDs provided in a text box below.

If a given user is not included in the cluster's ACL, this cluster will not appear in the cluster selection menu on any of the submission pages opened by this user – hence he/she will not be able to submit to this cluster explicitly. The cluster will also be excluded from consideration by the metascheduler when the "Auto" option is specified at submission time.

The cluster ACLs are very useful if a cluster has to be drained of BioHPC jobs or reserved for only a subset of users. The ACL information is stored in the table `clustersACL` in the `CBSUJOBS` database.

4.4 Application administration

The link **Manage applications** on the main administration page produces a list of all available applications and displays some of the attributes stored in table `apps` of the `CBSUJOBS` database. Attributes of a given application can be edited by using the corresponding **Edit** link in the last column. These attributes are:

- **Database ID**: unique application ID generated automatically when the application is first configured.
- **Name**: string used as a link to this application from the left bar of BioHPC page.
- **Description**: optional short description of the program.
- **Status**: Hidden (no link appears on the main BioHPC page – useful for temporarily hiding the application from the public), Internal (represented as an `aspx` web page within BioHPC – most applications have this status), External (application is a link pointing to a different web server), or Maintenance (application in this state will work only for administrators, others will get a maintenance info message). If the application is thrown into the Maintenance status, a window will appear below where an info message to the users may be entered; this message will appear

at the bottom of every BioHPC web page. The message will be cleared next time the application status changes.

- **Category**: determines the main software category under which this application will be listed on the right bar of BioHPC pages. Important for display purposes only. See **Manage Categories** later on.
- **Access**: Everyone (available to everyone, including the "guest" user) or Registered (available to registered user only). See User administration for more info.
- **Program**: name of the application executable (not used).
- **Web page**: name of the `aspx` file with the submission interface.
- **Max guest jobs**: maximum number of jobs with this application a guest user can have in cluster queues at any given time.
- **Max reg user jobs**: maximum number of jobs with this application a registered user can have in cluster queues at any given time.
- **Preferred cluster**: for jobs submitted with automatic cluster selection, this cluster will be considered first by the BioHPC built-in meta-scheduler; if not available, other clusters will be considered.
- **Logfile timeout**: if log file generated by the job is not updated for this number of minutes, the job will be reported as **stalled**. Leaving this field empty means that no check for stall condition will be performed.

Besides the main application attributes (stored in `apps` table), the administrator has to specify a number of cluster-dependent attributes. These are:

- **TIMEOUTS** (in minutes): determine how long this application will be allowed to run on a given cluster. If the timeout is exceeded, the job will be killed by the regardless of whether if completed or not. The timeouts may be different for guests and registered users. **If the timeout for a given cluster/user group is entered as zero, the application will not be allowed to run on this cluster for this user** (i.e., on the submission page, this cluster will not be listed in the cluster drop-down menu and it will not be considered when the "Auto" option is specified). Thus, **entering a zero timeout is the way to "turn off" a given application on a cluster**. The timeout information is stored in table `timeouts` of the `CBSUJOBS` database.
- **CPUs/node**: determines how many instances of the application executable may run concurrently on a single node of a given cluster. This parameter is only meaningful for parallel (or BioHPC-parallelized) applications, launched via MPI, when the **exclusive/node** flag is also checked (see below).
- **Exclusive/node**: if checked, the application will occupy exclusively the nodes it is running on, i.e., no other jobs will be dispatched to these nodes at the same time. If the application is an MPI program, the maximum number of MPI processes per node will be equal to the corresponding **CPUs/node** entry.
- **RAM/node, HD/node** : these fields are reserved for future development of the metascheduler; for the time being, they should be entered as "0".

Most applications can be safely submitted as non-exclusive (i.e. the **exclusive/node** flag should **not** be checked). Exceptions include cases where

- an application (or a single MPI process of a parallel program) cannot comfortably "coexist" with other applications because it needs to consume most of the node's memory and/or local scratch disk space,
- an application (or a single MPI process of a parallel program) is multi-threaded and able to consume all of the node's CPUs – running such an application in the company of others would impair its multi-threaded performance. Examples of such applications include P-BLAST, P-IPRSCAN, and P-HMMER.

The button **Manage Categories**, available from the main application administration page, allows the administrator to define new application categories, edit names of or remove the existing ones, and determine the order in which the main category buttons will be displayed on the left bar of the BioHPC web pages (the lower the priority number, the higher up the button will be located). Each application has to be assigned to one of the categories defined here (see description of the **Edit** link in the application table). The assignment is for display purposes only: the link to the application's `aspx` web page will appear under its category button when the latter is clicked.

New applications can be configured using the **Add new application** button located on the main application admin web page. This will generate a unique application ID in the `apps` table, and open a dialog where the main attributes and timeouts for the application can be entered. Please note that the **Add new application** operation merely configures a new application in the `CBSUJOBS` database. The main task associated with adding applications is to create a corresponding submission page and modifying a few other parts of the BioHPC code to work with this new page. In particular, one has to make sure that the entry of `enum tAppId` (in file `CbsuComm.cs`, project `CbsuComm`) corresponding to the new application has a numerical value matching the database ID of this application in the `apps` table. More guidelines for adding new applications will be found in the Section 7 (currently under construction).

Special applications. Two of the applications defined in the `apps` table, namely `blastdb` and `cleanJOBS`, have administrative purposes and can only be run by BioHPC administrators. They are configured as **Hidden**, so that they cannot be accessed through a public link. Administrative functions performed by these applications (updating BLAST databases and cleanup of old jobs data) require a lot of time and sizable computational resources, it is therefore important that they run as regular jobs on compute nodes of the cluster rather than directly on the web server, for example. Programming such resource-intensive administrative tasks as applications makes it possible to utilize standard job submission procedures and thus helps keep the code of BioHPC simple. More information about the two special applications can be found in Sections 4.2 and 4.6.

4.5 Site setup

The Site setup button provides access to the `setup.aspx` page which can be used to customize various basic parameters of BioHPC site through the file `web.config`. `setup.aspx` can only be open

on the web server where BioHPC is running. In fact, the page has to be run as a part of the installation process described in Section 3 and that's where the details are given.

4.6 Sequence database administration

Data-mining applications offered by BioHPC, such as BLAST and HMMER, work by comparing one or more query sequence (nucleotide or amino acid) to a set of known reference sequences. The reference sequences are collected in a (usually big) file, customarily referred to as a **sequence database**. It should be stressed, however, that such files are not databases in a sense of any commonly used database engine (such as MS SQL Server) or query language and should not be confused with the `CBSUJOBS` database holding the parameters of BioHPC and job attributes. From the point of view of BioHPC, the sequence databases discussed in this Section are just files, properly formatted and accessible only to the data-mining programs they have been designed for.

Sequence databases can be organized and subdivided using many different criteria, such as the type of organism, the part of the genome they represent, or the research project the reference sequences resulted from. This naturally leads to a multitude of database files, some of them public, and some, private, many frequently updated. Before submitting a BLAST of HMMER job, the user specifies which of the available sequence databases he/she wants to use. Before the job starts, all the requested database files must be available on the **local disks** of all the compute nodes the job will be running on. This local availability is very important if the network overhead (inevitable if the databases were to be stored on a network drive) is to be avoided. It therefore makes sense for the biggest and most frequently used databases to be replicated on all (or a subset of) compute nodes and stored permanently on local disks. Other databases may be stored in one copy, for example, on a network drive visible to all BioHPC clusters, and replicated on compute nodes as needed, right before the start of the job and only for the job's duration. BioHPC users have an option of using their own private databases, uploaded as FASTA files upon jobs submission and automatically formatted before being replicated on the nodes assigned to the job. If needed, such databases may be automatically stored on BioHPC file server for further use.

The button **Manage BLAST databases** available from the main **Administration** page takes us to the database administration web tool described in detail in this Section. Analysis of this tool reveals the principles behind the sequence database organization in BioHPC. The page offers are a few management buttons, each corresponding to a specific type of a database:

Core databases: denotes standard files publicly available directly from the publishers (e.g. nr from NCBI) and updated frequently. These files need to be replicated on local disks of all compute the nodes. Clicking on the **Core databases** button will display statistics about the files currently available on different clusters, with dates of last propagation and the date of last download from the publishers' web/ftp sites. The button **Manage** will display an editable table where basic parameters of various file can be configured (including names and publishers' URL), new databases can be defined, and the ones which are no longer needed can be deleted from the configuration. Clicking on Submit Changes (under the table) will modify the appropriate entries in the **sequence database configuration table** `dbfiles`

in the CBSUJOBS database, but it will not modify the actual sequence database files. To download current versions of the database files from the publishers (as configured in dbfiles table), click on **Download and format** button. This will queue up a job (on a cluster selected from the drop-down list below) executing a perl script update_blastdb.pl. The script will download the proper files, unzip them if needed, format the FASTA files (using the standard formatdb program from the BLAST distribution), and place the formatted files in the network directory H:\CBSU\blastdb\<current> on the file server, where <current> denotes today's date in the *yyyymmdd* format (e.g., 20070828). A file current containing this date will also be created in H:\CBSU\blastdb. Once the files are downloaded, formatted, and placed in H:\CBSU\blastdb\<current>, they need to be propagated to the nodes. This is accomplished by clicking on the **Propagate** button, which will queue a job on the cluster selected for propagation. Once the job starts on one of the compute nodes of this cluster, it will execute a program called propagate.exe which will fork into several threads as specified in the "concurrent nodes" text box at the time of submission. Each such thread will update one node of the cluster, and then will get "reused" to update another one, and so on, until all nodes are eventually updated. At the end of the operation, the directory local T:\CBSU\blastdb\<current> on each node will be synchronized with the central storage H:\CBSU\blastdb\<current>. The synchronization process is quite intelligent – it does not touch files already in sync between the two directories, and the old <current> subdirectory will be removed from the local storage.

Additional databases are stored locally on the nodes, don't change too often and sometimes require special processing. They have to be manually downloaded and formatted (if needed) and then placed in H:\CBSU\blastdata directory on the file server, from where they can be propagated to the nodes. HMMER databases (Pfam) are in this category. Clicking on the **Additional databases** button will display statistics about the files currently available on different clusters with dates of last propagation. The button **Manage** will display an editable table where names and types of various files can be configured, new databases can be defined, and the ones which are no longer needed can be deleted from the configuration. Clicking on **Submit Changes** (under the table) will modify the appropriate entries in the **sequence database configuration table** dbfiles in the CBSUJOBS database, but will not modify any actual sequence database files. Once these files are manually placed in H:\CBSU\blastdata directory on the file server, they need to be propagated over the nodes. This is accomplished by clicking on the **Propagate** button, which will queue a job on the cluster selected for propagation. Once the job starts on one of the compute nodes of this cluster, it will execute a program called propagate.exe which will fork into several threads as specified in the "concurrent" nodes text box at the time of submission. Each such thread will update one node of the cluster, and then will get "reused" until all nodes are eventually updated. At the end of the operation, the directory local T:\CBSU\blastdata on each node will be synchronized with the central storage H:\CBSU\blastdata. The synchronization process is quite intelligent – it does not touch files already in sync between the two directories.

Private databases are added by the users at the time of job submission. They are stored (in formatted form) on the network drive, typically in directories H:\CBSU\blastdata1,

H:\CBSU\blastdata2, or in subdirectories thereof. Local copies may be stored locally on real-time nodes. The **Manage** button will display a table, where all the attributes of the private databases can be edited. Besides the standard attributes (names, type, location on H:\CBSU) one can also specify whether or not to keep a local copy on the nodes of the RT cluster as well as the list of registered users allowed access to a given file. Clicking on **Submit Changes** (under the table) will modify the appropriate entries in the **sequence database configuration table** dbfiles in the CBSUJOBS database, but will not modify any actual sequence database files. Also, records will be added to or removed from the table userdbaccess, which controls access of registered users to various private databases. In order to propagate the private database files to the RT compute nodes, use the button **Synchronize local copies of selected databases on RT nodes** (this action will also propagate selected "Global" databases – see below). A job will then be queued on the RT cluster to execute a program called syncRT.exe. Upon completion of this job, the directory local T:\CBSU\blastdata1 on each node of the RT cluster will contain all private and global databases specified to be propagated. The synchronization process is quite intelligent – it does not touch files already in sync between the local and the shared directories. Note that the user access to private sequence databases can also be handled from the **Manage users** page.

Global databases are stored on the network drive, don't change too often and sometimes require special processing. They have to be manually downloaded and formatted (if needed) and then placed in their respective subdirectories on H:\CBSU. Local copies may be stored locally on real-time nodes. The **Manage** button will display a table, where all the attributes of the global databases can be edited. The attributes include the names of the database and the corresponding file, sequence type (nucleotide or protein), the directory under H:\CBSU where these files are located, and the category to which a given database belongs. Categories of global databases are introduced for organizational purposes only and can be managed using the **Manage categories** button. For each database, one can also specify whether or not to keep a local copy on the nodes of the RT cluster as well as whether the file will be available to all users or only to selected ones. In the latter case, the access privileges have to be configured on the user level, using the **Manage users** page. Clicking on **Submit Changes** (under the table) will modify the appropriate entries in the **sequence database configuration table** dbfiles in the CBSUJOBS database, but will not modify any actual sequence database files. Also, records will be added to or removed from the table userdbaccess, which controls access of registered users to various private databases. In order to propagate the private database files to the RT compute nodes, use the button **Synchronize local copies of selected databases on RT nodes** (this action will also propagate selected private databases). A job will then be queued on the RT cluster to execute a program called syncRT.exe. Upon completion of this job, the directory local T:\CBSU\blastdata1 on each node of the RT cluster will contain all private and global databases specified to be propagated. The synchronization process is quite intelligent – it does not touch files already in sync between the local and the shared directories.

A record of sequence database updates on compute nodes is kept in table dblog in the CBSUJOBS database. This table is updated every time a propagation job is submitted and then again when this job ends. The update statistics/history information displayed by the database administration tool is read off this table.

5 Real-Time Clusters

Some jobs submitted to BioHPC are short, one-processor tasks with simple input and output. For example, sequence retrieval from a database or a quick BLAST analysis of a few query sequences fall into this category. BioHPC offers a possibility to process jobs like these in real time (as opposed to the regular batch mode used for more time consuming jobs) on a separate dedicated cluster, referred to as RT (Real Time) cluster. An RT cluster is also required for the personal database registration functionality of BioHPC's P-BLAST application.

Similarly to the "regular" clusters, an RT cluster is a set of nodes under control of a cluster scheduler. While any of the available schedulers could be used in principle (for example, CCS), BioHPC offers its own scheduler, referred to as the **RT scheduler**, implemented using MS SQL database and the standard `jt.exe` windows task scheduling tool.

An RT cluster must consist of at least one Windows Server 2003 or Windows Server 2008 machine satisfying the following conditions:

- be in the same Windows domain as BioHPC web server and have access to the common network share `H:\` (although this share does not have to be permanently mounted as `H:\`)
- have a local scratch disk assigned as `T:\`, with the directory `T:\CBSU`
- the BioHPC IDU must have administrator privileges
- utility programs `runrt.bat`, `rtexe.exe`, `RTclean.exe` (all parts of BioHPC) must be available in `T:\CBSU`

The BioHPC RT scheduler consists of the following components:

- Two tables in `CBSUJOBS` database: `rtnodes` (holds information about the nodes comprising the RT cluster) and `rtjobs` (holds information about jobs submitted to the RT cluster).
- The program `RTsched.exe` (included in BioHPC - the actual scheduler) running on the web server as a scheduled task, started at server boot time and running continuously thereafter with IDU credentials.
- The `jt.exe` task scheduling tool (Microsoft Task Scheduler Command Line Utility) in `H:\CBSU` network directory.
- `runrt.bat`, `rtexe.exe`, `RTclean.exe` – utility programs stored in local `T:\CBSU` directory on each of the RT cluster nodes.
- A set of APIs to communicate with the RT scheduler form within the BioHPC web pages (a part of BioHPC).
- RT cluster administration application (analogous to the Compute Cluster Administrator in CCS); this is implemented as a web page available from the main BioHPC administration page by clicking on the **Real-Time Scheduler** button. It provides basic functionality for monitoring of the RT compute nodes and control of the jobs submitted to RT cluster at the cluster level (rather than BioHPC level).

The RT APIs called from within BioHPC interact with the scheduler `RTsched.exe` via the database table `rtjobs`. For example, submitting a job through an API function is equivalent to adding a new line to `rtjobs` containing, among other entries, the unique scheduler job ID of the new job (not to be confused with the BioHPC job ID assigned by BioHPC) and a flag describing this new job as queued up and waiting. Likewise, the `rtjobs` table is consulted by the APIs when RT jobs are to be listed, or when an RT job needs to be canceled. The role of the scheduler process `RTsched.exe` is to periodically (every 30 seconds) examine the `rtjobs` table and reconcile its content with what is happening on the RT compute nodes. Thus, for a job flagged as waiting, `RTsched.exe` will checks for RT node availability and send this job to one of the free nodes if such are found and then update the job's flag in `rtjobs` to "running". On the other hand, a job flagged in `rtjobs` table as "running" will be looked for on the compute node and if is not found there any longer, its status in `rtjobs` will be updated to "finished". While performing any job-related actions on the compute nodes, such as job submission, listing, or killing, the `RTsched.exe` program contacts the compute nodes via the `jt.exe` program, i.e., it remotely schedules various "scheduled tasks" to be executed on these nodes. This is why the BioHPC's IDU has to have administrator privileges on the compute nodes.

6 Remote Clusters

BioHPC is equipped with mechanisms which allow submission of jobs to remote clusters located in different domains than the rest of the BioHPC infrastructure. While handling of remote clusters is integrated rather seamlessly in BioHPC both from the users' and from the administrator's perspectives, there are a few specific issues here that the administrator should be aware of. This Section describes how these issues are addressed in BioHPC.

In essence, a remote cluster differs from a local one in two basic aspects. First, the cluster scheduler is not directly accessible by the BioHPC's Interface Domain User (IDU) and therefore jobs cannot be submitted and controlled using the standard set of scheduler APIs. This problem is solved by using the web service-based job submission system based on the Job Submission Description Language (JSDL) which has to be deployed on a remote cluster. BioHPC features wrappers around JSDL functions providing syntax and functionality mimicking those of the local scheduler APIs. Second, the fact that the remote cluster is in a different network and Windows domain than BioHPC means that the remote compute nodes have no direct access to the BioHPC's job directories. Therefore, jobs submitted to a remote cluster cannot read the input data from or deposit output data directly to the domain that BioHPC is in. To address this problem, BioHPC uses the ftp protocol to transfer jobs' data back and forth between the local and remote domains.

The rest of this Section provides more specifics on how BioHPC handles remote clusters.

In order to be compatible with BioHPC, a remote cluster must satisfy the following conditions:

- The cluster must be running Windows Server 2003 with Compute Cluster Server (CCS) pack or Windows HPC Server 2008 with the native MPI library and scheduler.

- The JSDL-based job submission web service has to be deployed (it comes standard with HPC Server 2008); the URL of this web service (referred to as **JSDLAddress**) has to be known (something like https://host.your-address.com/myjsdlcluster.svc)
- There must be a dedicated domain account, such as `domain\username` (referred to as **JSDLUserName** with password **JSDLUserPass**) on the domain of the remote cluster; BioHPC will use this account to connect to the cluster via the JSDL web services and ftp (see below).
- Similarly as in the case of local clusters, there must be a network share (referred to as **JSDLdir**) accessible from all nodes of the remote cluster. A UNC path (such as \\fileserver\share\biohpc) must be available. This directory will play a role analogous to the `H:\` drive on local clusters, i.e., it will host the subdirectories `CBSU`, `CBSU\JOBS`, as well as the binaries and ptools directories. All the binaries of the applications and tools will have to be copied to these directories.
- There must be an ftp server, e.g., `ftp.your-address.com` (referred to as **JSDLftpServer**), accessible to the user **JSDLUserName**. The **JSDLdir** has to be accessible for writing via this ftp server.

Let's take a closer look at how various BioHPC functions handle remote clusters. As mentioned earlier, all mechanisms are completely transparent to users and administrators alike. The explanations below are provided mainly for completeness and better understanding of BioHPC design.

Job submission. As with local clusters, the job directory is first prepared locally in BioHPC, containing all the user input data and the `run.bat` script. The remote cluster submission API packs the content of the job directory into a zip archive and then uses ftp to create a corresponding job directory on the remote cluster (specifically, as a subdirectory of **JSDLdir**`\CBSU\JOBS`) and transfer this archive, along with `run.bat`, to that directory. The job (i.e., `run.bat`) is then submitted on the remote cluster via JSDL web service in the remote job directory and the database status of the job is set to QUEUED. Once the job starts executing, one of the first commands in `run.bat` (specially constructed for a remote cluster) will unpack the zip archive, extracting the actual input data into the remote job directory, from where it will be picked up by the compute nodes assigned to the job on the remote cluster.

Keeping track of job status. As on the local clusters, the program `cbsufinalize.exe` is invoked in the beginning of `run.bat` with indicator "0", signaling that the job just started. Normally, this program calls up the `cbsufinalize.aspx` web page (a part of BioHPC) which changes the database status of the job to RUNNING. In the case this web page cannot be accessed from the remote cluster, `cbsufinalize.exe` will just create the file *jobID*`.STATUS.0` in the remote job directory. The presence of this file will indicate (to anyone who looks) that the job has started. This file will be later detected by the utility daemon `JSDLupdate.exe`, running periodically on the BioHPC web server and scanning all the remote job directories through ftp (see description of `JSDLupdate.exe` below). If the file *jobID*`.STATUS.0` is detected, `JSDLupdate.exe` will call `cbsufinalize.aspx` (more specifically – a public method exposed from that page) to update the job's database status to RUNNING. All subsequent changes in the database status of a remote job as well as output data transfer from finished remote jobs are also handled by `JSDLupdate.exe`.

JSDLupdate.exe. This utility program is a part of BioHPC. It handles changes to database job status of remote jobs, imposition of timeouts on remote jobs, and transfer of output data from finished (including canceled and failed) remote jobs. It is scheduled to run periodically on the web server. During each run, the program scans (via ftp) all active remote job directories on remote clusters and for each such job it performs the following actions:

- If the current database job status of this job is QUEUED, the file *fileID*.STATUS.0 is present, and the job is reported as "in queue" by the remote scheduler, then database job status is changed to RUNNING.
- If the current database job status of this job is RUNNING and the job is reported as "in queue" by the remote scheduler, then JSDLupdate.exe checks whether the job exceeded the timeout; if so, it sends the "cancel" command to the remote scheduler, terminating the job, and initiates the transfer of output data.
- If the current database job status of this job is RUNNING and the job is reported as "out of queue" by the remote scheduler, then JSDLupdate.exe attempts to transfer the output data. If the transfer operation is successful and finished, the database job status is changed to an appropriate "finished" state using a call to the cbsufinalize.aspx page and the remote job directory is deleted. A completed data transfer is indicated by the creation of the file *jobID*.filetransfer.done in the local (i.e., in the BioHPC domain) job directory. If the data transfer was not finished or failed, this file will not be created and the database job status will remain RUNNING. In such a case, JSDLupdate.exe will come back to this job when it runs next time and attempt the output transfer again.
- If the current database job status indicates one of the "finished" states, but the local job directory does not contain the file *jobID*.filetransfer.done, JSDLupdate.exe attempts to finish off the output transfer.

Details of output data transfer through JSDLupdate.exe. The idea behind the transfer of output data from remote jobs is to first create a zip archive (called *jobID*dataout.zip) of the remote job directory, transfer it via ftp to the local job directory, and unzip it there. When JSDLupdate.exe first detects (by querying the remote scheduler) that a remote job has ended, it initiates the output transfer process by scheduling the *jobID*dataout.zip archive to be created. This scheduling is done in one of two different ways, configurable through setup.aspx:

- By submitting a special "zipping" script makezip.bat (provided in BioHPC distribution) as a job to the remote cluster queue. The script must be present in the remote directory **JSDLdir**\CBSU\JOBS. This is the only option when the BioHPC administrators do not have administrative access to the remote cluster. If the remote cluster is busy, this "zipping" job may have to wait in queue before it is executed.
- Using the simple zipdaemon.pl utility program provided (currently implemented in perl), scheduled to run periodically on the remote cluster and call makezip.bat for all jobs for which entries have been created by JSDLupdate.exe in the remote directory **JSDLdir**\CBSU\JOBS\zipdaemon. This option is preferred, but available only on those

remote clusters where the BioHPC administrators have administrative access (to be able to configure `zipdaemon.pl` as a scheduled task).

Having scheduled the creation of the archive, `JSDLupdate.exe` leaves a file called *jobID*`zip.dispatched` in the local job directory to mark this fact, which completes the processing of this job in this round of `JSDLupdate.exe`. The daemon will return to this job at its next invocation – this time it will check if the zip archive has already been created. If not (and usually it takes some time), the daemon will leave the job alone until next time. If the archive is there and is detected as complete (a *jobID*`zip.DONE` file is created in the remote job directory after a successful zipping), the daemon will transfer it via ftp to the local job directory, unzip it there, mark the successful transfer with the *jobID*.`filetransfer.done` file, delete the remote job directory, and change the database job status to the appropriate "finished" state through the `cbsufinalize.aspx` function. Within this general scheme, `JSDLupdate.exe` is able to detect various error conditions that may occur during the zipping/transfer process and correct them by rescheduling the zip file creation from scratch. **When the output data transfer is complete, all the output files are available in the local job directory and can be accessed by the users in a normal way using the `showfile.aspx` web page.**

Job monitoring. Monitoring and control of remote jobs during execution is accomplished in exactly the same way as it is done for local jobs, that is, using the `showfile.aspx` and `cbsujob.aspx` web pages, typically pointed to by links in the notification e-mails. These pages are "aware" of the fact that in the case of remote jobs any files they attempt to retrieve are located in remote job directories and need to be transferred via ftp before being presented to the user. Thus, both job control tools are completely transparent, although accessing remote files generally takes longer than accessing local files.

7 PROGRAMMER'S GUIDE

[UNDER CONSTRUCTION]

8 APPENDIX

[UNDER CONSTRUCTION]

8.1 Structure of `run.bat` script

[UNDER CONSTRUCTION]

8.2 `CBSUJOBS` database

[UNDER CONSTRUCTION]

8.3 Programs and what they do

[UNDER CONSTRUCTION]