STORK PROJECT

VERSION 2.0

User Manual v-2.0

Stork Project Team
Center for Computation and Technology
Louisiana State University

November 10, 2010

Contents

Chapter 1

Introduction to Stork

1.1 Welcome to Stork

Welcome to the Stork version 2.0 user manual. Stork is developed by Stork project team at the Center for Computation Technology in Louisiana State University (LSU) and was first developed as a part of Condor team in 2001 by Dr. Tevfik Kosar. Stork have grown tremendously in these years to become a stand alone software that is being currently used by many industries, academic and research institutions all around the globe. Stork is considered one of the very first examples of data placement and scheduling and optimization tools. We hope that Stork will help you to provide a better solution in terms of data bottleneck in your computing environments.

1.2 Introduction

Stork is a batch scheduler specialized in data placement and data movement, which is based on the concept and idea of making data placement a first class entity in a distributed computing environment. Stork understands the semantics and characteristics of data placement tasks and implement techniques specific to queuing, scheduling, and optimization of these type of tasks.

Stork acts like an I/O control system (IOCS) between the user applications, the underlying protocols and data storage servers. It provides complete modularity and extendibility. The users can add support for their favorite storage system, data transport protocol, or middleware very easily. If the transfer protocol specified in the job description file fails for some reason, Stork can automatically switch to any alternative protocols available between the same source and destination hosts and complete the transfer.

Stork can interact with higher level planners and workflow managers. This allows the users to schedule both CPU resources and storage resources together. Currently, some implementations of Condor DAGMan and Pegasus come with Stork support.

1.3 Availability

Stork is currently available as a free download for it's entire user base at URL http://stork.cct.lsu.edu/downloads/ . Both source code and Binary distributions of stork version 2.0 are available for the platforms detailed in the appendix section. A platform is an architecture/operating system combination. Stork binaries are available for most major versions of UNIX and MAC OS. All the versions and distributions were build and tested in the NMI test bed at UW madison.

1.4 Privacy Notice

The Stork software will send short messages that includes the domain name, IP address and OS name to the Stork project team at LSU. This information collected is to generate reports to publish the use of Stork project world wide. This information will help the Stork team to improve the quality of software accordingly.

The Stork team will not use these reports or any information collected from the usage of stork to publicly identify any stork system or users with out their permission. No personal information is collected from the users.

Users can also disable this feature if they don't want to send any information back to the Stork team by changing the enable stat report to false in stork config file.

1.5 Contact Information

Latest software releases, news about Stork or any publications that have resulted in Stork research can be found at the Stork website which is http://www.stork.cct.lsu.edu.

In addition, Stork project maintains two e-mail lists for it's users. One is stork-announce@cct.lsu.edu is a low-traffic list and archive used solely for announcements of software releases and bug fixes. The other one is stork-discuss@cct.lsu.edu is a list and archive for general questions and discussion about using and applying Stork.

Users can subscribe to any of these lists by registering at our website. Users can feel free to subscribe to either, or search the list archives. You must be subscribed to a list before you can post messages.

1.6 Stork Team

Stork team here at Louisiana State University is headed by Dr. Tevfik Kosar (kosar@cct.lsu.edu) who holds an joint appointment with department of Computer science and Center for Computation and technology. He is the primary designer and architect of Stork project. Sivakumar Kulasekaran (sivakumar@cct.lsu.edu) is working for the Stork team as an Information Analyst.

Many graduate and undergraduate students are also part of the Stork team. Some of the team members are Brandon Ross, Sivakarthik Natesan, Esma Yildirim, Dengpan Yin, Georgi Stoyanov.

1.7 Acknowledgements

Stork team likes to acknowledge National Science Foundation (NSF) for sponsoring this project under award numbers OCI-0926701. Also, we would like to thank Center for Computation and Technology at LSU for their facilities, LONI network for their computational resources.

1.8 Copy right

Chapter 2

Building and Installing Stork

2.1 Installation from Binary Package

Stork can be downloaded from the Stork project homepage by clicking downloads then followed by clicking Stork Binary compilation release. After downloading, simply extract the package with whatever tools you have at your disposal for decompressing and untarring gzipped tar archives. Most users would simply use the tar command. For example, to extract the Stork 2.0 binaries:

```
$ tar zxvf stork-2.0.tar.gz
```

This will create a directory in your current directory containing the binaries. Afterwards, simply `cd` into the directory.

2.1.1 Installation

For installing Stork from binary compilation, you simply have to run the included installation script with the desired installation path *<install-path>* as an argument: `./stork-install.ksh` *<install-path>*

This will extract the included binaries and place them in the location specified by *<install-path>*.

2.1.2 Post-installation

Now that you have installed Stork, you must configure your environment in order to use Stork with minimal effort. After your installation process of Stork completes, you will be shown a message asking you to set your `PATH` and `STORK_CONFIG` environment variables by running commands similar to these:

```
$ export STORK_CONFIG=/path/to/stork/etc/stork_config
$ export PATH=$PATH:/path/to/stork/bin:/path/to/stork/sbin
```

You should do as the script says! In order to make sure that your environment is configured properly every time you start a new shell, you should also add these commands to a start-up script (e.g., the `.bashrc` file in your home directory). This will prevent you from needing to run these commands every time you want to use Stork.

If you have reached this point without a problem, you now have a working Stork installation. :) You may test this by starting a Stork server with the command `stork_server`. If you would like to edit the `stork_config` file and configure your Stork installation further, it may be found in the `etc` directory inside the newly installed Stork directory.

2.2 Installation from Source

This section is for users who wants to build Stork from the source code and for users who needs to customize their external third party transfer modules with their installation. Users who have compiled Stork using binary package may skip this section.Stork may be downloaded from the Stork project homepage by clicking downloads then followed by clicking Stork source code release.

If you are a user of a UNIX-like system, it is recommended that you download the source code, as this will allow you to better customize your installation. If

you are running on a system on which Stork is not known to compile, or you find you are not able to compile from source for other reasons, you may want to see if a binary package is available for your system. This information can be obtained by viewing Stork supported platforms.

After downloading, simply extract the package with whatever tools you have at your disposal for decompressing and untarring gzipped tar archives. Most users would simply use the tar command. For example, to extract the Stork 2.0 source:

```
$ tar xvf stork-2.0.tar.gz
```

This will create a directory in your current directory containing the source code. Afterwards, simply `cd` into the directory.

2.2.1 Building third party transfer modules

If you plan on building contributed (third-party) transfer modules for Stork which is optional for the users who just need the core Stork functionalities, you will have to download the externals package from the Stork website, and place its contents into the `./externals/bundles/` directory. If you have no such plans, you may skip this step completely.

The externals package, at this point, contains the following bundles: globus, irods, openssl, SRB and SRM. The following table shows the protocols with available transfer modules and the bundles required to compile them:

Protocol	Bundles
FTP	globus, openssl
GSIFTP	globus, openssl
HTTP	globus, openssl
iRODS	irods, srb
SRB	srb
SRM	srm
Petashare	irods, srb

Please note that Stork uses Globus components for file transfers using the *GSIFTP* protocol (GridFTP). Therefore, you need to have the Globus Toolkit installed and configured on your system with user/host certificates in order to use this transfer module.

2.2.2 Configuring

After downloading and extracting the source, you must configure the source tree with the `configure` script therein. Note, however, that if you are trying to build a development version of Stork (odd minor version numbers), then you will need to have `autoconf` and `autoheader` (at least version 2.59) installed in order to first generate the `configure` script. This is unnecessary if you are trying to build a release, as releases already contain a `configure` script.

To build the Stork client commands, the server, and the core (supported) transfer modules, and install to the default `/usr/local/stork` directory, simply run `./configure` with no options. However, if you want to build contributed (third-party) transfer modules, want to build Stork with other features, or need to specify a special compiler or library to use, you must use additional options with `./configure`. (A complete list of options may be seen by running `./configure --help`.)

You will probably want to specify a location other than the default location (`/usr/local/stork`) to install Stork. If you are not a superuser, this is a necessity. This can be done by specifying `--prefix=`*`path`* with `./configure`, where *`path`* is the directory in which you want to install Stork.

You might also want to build contributed transfer modules along with Stork. This can be done by specifying `--with-`*`bundle`* with `./configure`, where *`bundle`* is the name of the external package necessary to build the transfer module in question. If you specify *all* of the necessary bundles for a given transfer module (as specified in the table in the Externals section) with `--with-`*`bundle`*, and the bundles can all be found in the `./externals/bundles/` directory, then the transfer module in question will be built. More information about this can be found in the Externals section above and by running `./configure --help`.

For example, to build all transfer modules configure file looks as follows

./configure –prefix=path (where you want to install) –with-openssl –with-globus –with-srb –with-srm –with-irods –with-petashare

`configure` will take a few minutes to check that everything necessary for building Stork is available on your system. After it is done running, you are ready to move on to the next step.

2.2.3 Building and installing

If you have been building from source up to this point, all you have to do now is run `make install or make` . This will install everything to the location specified with `--prefix` when you ran `./configure`, or to `/usr/local/stork` if you did not specify anything. After it has installed (it shouldn't take long), you may go to the next step.

Depending on the speed of your system and the number of externals you require, building may take anywhere from 3 to 20 minutes. Just be patient.

Note that while external modules are building, output will stop for a significant amount of time. Keep in mind that this is normal, and don't worry that your screen may have frozen. If you are interested in watching the output from those external builds for some reason, you may do so by running (in another terminal, of course) `tail -f` with the path to the build log shown before the module begins building given as an argument.

2.2.4 Post-installation

Now that you have installed Stork, you must configure your environment in order to use Stork with minimal effort. After your build process of Stork from source completes, you will be shown a message asking you to set your `PATH` and `STORK_CONFIG` environment variables by running commands similar to these:

```
$ export STORK_CONFIG=/path/to/stork/etc/stork_config
$ export PATH=$PATH:/path/to/stork/bin:/path/to/stork/sbin
```

You should do as the script says! In order to make sure that your environment is configured properly every time you start a new shell, you should also add these commands to a start-up script (e.g., the `.bashrc` file in your home directory). This will prevent you from needing to run these commands every time you want to use Stork.

If you have reached this point without a problem, you now have a working Stork installation. :) You may test this by starting a Stork server with the command `stork_server`. If you would like to edit the `stork_config` file and configure your Stork installation further, it may be found in the `etc` directory inside the newly installed Stork directory.

Chapter 3

Using Stork

3.1 Starting Stork Server

The Stork server is the main component of the Stork scheduler. The Stork server runs as a persistent daemon process and performs management, scheduling, execution, and monitoring of data placement activities.

The Stork server accepts the following parameters (defined by `STORK_ARGS`)

```
$ sbin/stork_server --help
============================================================
USAGE: stork_server
   [ -t           ] // output to stdin
   [ -p           ] // port on which to run Stork Server
   [ -help        ] // stork help screen
   [ -Config      ] // stork config file
   [ -Serverlog   ] // stork Server log in ClassAds
   [ -Xmllog      ] // stork user log in XML format
============================================================
```

If a user called `stork` exists, the server will switch to the `stork` user for security purposes.

The command below starts the Stork server connected to port `<port>` The stork logs are named with the prefix Stork, such as StorkLog, Stork.history, etc.

```
stork_server -p <port>
```

The Stork server generates a log file which is used for logging its activities.

Below is a sample of the log file:

```
cat local/log/StorkLog
11/3 15:04:54 ******************************************************
11/3 15:04:54 ** stork_server (STORK) STARTING UP
11/3 15:04:54 ** /home/sivahpc/storkie/stork-2.0/stork/stork/sbin/stork_server
11/3 15:04:54 ** SubsystemInfo: name=STORK type=DAEMON(10) class=DAEMON(1)
11/3 15:04:54 ** Configuration: subsystem:STORK local:<NONE> class:DAEMON
11/3 15:04:54 ** $StorkVersion: 1.2.0 Nov  3 2010 $
11/3 15:04:54 ** $StorkPlatform: X86_64-LINUX_FC6 $
11/3 15:04:54 ** PID = 1167
11/3 15:04:54 ** Log last touched 11/3 15:02:41
11/3 15:04:54 ******************************************************
11/3 15:04:54 Using config source: /home/sivahpc/storkie/stork-1.3.0/stork/stork/etc/stork_config
11/3 15:04:54 DaemonCore: Command Socket at <130.39.225.155:40411>
11/3 15:04:54 Warning: Collector information was not found in the configuration file. ClassAds will not be sent to the collector
11/3 15:04:54 DaP log file: XML log file: %s

11/3 15:04:54 Submitting anonymous usage report to stork.cct.lsu.edu
11/3 15:04:54 MAX_NUM_JOBS = 10
11/3 15:04:54 MAX_RETRY = 10
11/3 15:04:54 MAXDELAY_INMINUTES = 10
11/3 15:04:54 VERIFY_CHECKSUM = FALSE
11/3 15:04:54 VERIFY_FILESIZE = FALSE
11/3 15:04:54 CHECKPOINT_TRANSFER = FALSE
11/3 15:04:54 SET_PARALLELISM = 1
11/3 15:04:54 MODULE_DIR = /home/stork//libexec
11/3 15:04:54 TMP_DIR = /home/stork//tmp
11/3 15:04:54 modules will execute in /home/stork//tmp
11/3 15:04:54 TMP_CRED_DIR = /home/stork//tmp
11/3 15:04:54 LOG = /home/stork//log
.....................
```

3.2 Stork Client Tools

3.2.1 Stork Submit

`stork_submit` is a client-side tool used to submit stork jobs to the Stork server.

```
$ bin/stork_submit
Usage: stork_submit [option]... [stork_server[:port]] submit_file
  stork_server[:port]
               specify Stork server host/port (deprecated)
  submit_file  path to Stork submit file

Options:
  -l <note>    include note in userlog (used by DAGMan)
  -stdin       read submission from stdin instead of a file
  -help        print this help information
```

```
  -version     print version information
  -debug       print debugging information to console
  -name <host[:port]>
               specify Stork server host/port
```

You can submit job to stork server using stork submit as follows. In this example, dapfile.dap is the dap file that is submitted to the server.

```
[sivahpc@dsl-condor bin]$ ./stork_submit dapfile.dap

Stork server will display the following output to conform your submission

using default proxy: /tmp/x509up_u24489
================
Sending request:
    [
        dest_url = "gsiftp://oliver1.loni.org/work/siva/dsanju";
        arguments = "-s 200M";
        src_url = "gsiftp://eric1.loni.org/work/siva/18GB";
        err = "new.err";
        output = "new.out";
        dap_type = "transfer";
        x509proxy = "/tmp/x509up_u24489"
    ]
================

Request assigned id: 16
```

3.2.2 Stork Status

`stork_status` is a client side tool used to query regarding the status of jobs submitted to the Stork server.

```
$ bin/stork_status
Usage: stork_status [option]... [stork_server[:port]] job_id
  stork_server[:port]
               specify Stork server host/port (deprecated)
  job_id       id of the job whose status you want

Options:
  -help        print this help information
  -version     print version information
  -debug       print debugging information to console
  -name <host[:port]>
               specify Stork server host/port
```

The dap_id is used by the stork_status commmand to query the Stork server. The dap_id is generated and assigned to a job when it is submitted to Stork using the stork_submit command.

The stork_status command accepts the following parameters, where host_name is optional. The host_name is used to specify a Stork server on a remote host.

Stork submit will produce two different outputs depending upon when it is called. If it is called before the process completes, sample output is shown below

```
[sivahpc@dsl-condor bin]$ ./stork_status 15
===============
status history:
===============


    [
        execute_host = "<130.39.225.155:48883>";
        dest_url = "gsiftp://oliver1.loni.org/work/siva/dsanju";
        arguments = "-s 200M";
        src_url = "gsiftp://eric1.loni.org/work/siva/18GB";
        remote_user = "sivahpc@dsl-condor.csc.lsu.edu";
        status = "request_received";
        err = "new.err";
        pid =  -1;
        dap_id = 15;
        output = "new.out";
        dap_type = "transfer";
        owner = "sivahpc";
        submit_host = "<130.39.225.155:34796>";
        x509proxy = "/tmp/cred-15";
        timestamp = absTime("2010-09-21T16:03:13-0500");
        use_protocol = 0
    ]
```

If the job is completed by the stork server, then the status for the particular job will be as shown below

```
[sivahpc@dsl-condor bin]$ stork_status 10


===============
status history:
===============


    [
        status = "request_completed";
        dap_id = 10;
        timestamp = absTime("2010-09-21T12:11:43-0500")
    ]

===============
```

3.2.3 Stork Log

`stork_log` is a client side tool used to bring the user log from the stork server to the client's location.

The `stork_log` command accepts the following parameters, where `host_name` is optional. The `host_name` is used to specify a Stork server on a remote host.

```
./stork_log [-h|--help] [-p pid] <command> [args] ...

A tool to transfer log files for Stork jobs from a Stork server to the
local machine.

Options:
  -h, --help  Print this usage information and exit
  -p <pid>    Specify the pid of the stork_log daemon to issue commands to

Valid commands and associated parameters are:
  get <host[:port]> <dap_id> <file_path>
    Retrieve a userlog for job with given dap_id from the Stork server at
    host[:port], and place it at file_path.
```

3.2.4 Stork Remove

`stork_rm` is a client side tool used to delete any jobs that are currently queued with the Stork server.

```
./stork_rm -h
usage: stork_rm  [option]... [stork_server] job_id
stork_server                    specify explicit stork server (deprecated)
job_id                          stork job id
       -help                   print this help information
       -version                print version information
       -debug                  print debugging information to console
       -name stork_server      stork server
```

Stork remove can be called as follows. Here 16 is the job id

```
[sivahpc@dsl-condor bin]$ ./stork_rm 16
Stork job 16 has been removed from the queue.
```

You can check the status of this job by calling stork status. The output will be then

```
[sivahpc@dsl-condor bin]$ ./stork_status 16
===============
status history:
===============


    [
        status = "request_removed";
        dap_id = 16;
        error_code = "REMOVED!";
        timestamp = absTime("2010-09-21T16:09:53-0500")
    ]
```

3.2.5 Stork Queue

`stork_q` is a client side tool used to retrieve a listing of jobs that are currently queued with the Stork server.

```
./stork_q -h
Usage: stork_q [option]... [stork_server[:port]]
  stork_server[:port]
               specify Stork server host/port (deprecated)

Options:
```

```
-help        print this help information
-version     print version information
-debug       print debugging information to console
-name <host[:port]>
             specify Stork server host/port
```

Sample output from the `stork_q` command:

```
[
    dest_url = "file:/home/user1/stork/data10M_48";
    src_url = "file:/home/user1/stork/data10M";
    remote_user = "user1@dsl-turtle06.cct.lsu.edu";
    status = "request_rescheduled";
    dap_id = 264;
    use_protocol = 0;
    stork_server = "qb1.loni.org";
    dap_type = "transfer";
    error_code = "port not accessible";
    num_attempts = 1;
    owner = "user1";
    cluster_id = 264;
    timestamp = absTime("2008-05-28T14:52:22-0500");
    generic_event = "Rescheduling."
]
```

Stork server runs as a persistent daemon process. It constantly listens to requests from the clients. The clients send their requests to the Stork server over the network using `stork_submit` command line tool in form of a ClassAd (Classified Advertisement).

Since Stork is designed to work in a heterogeneous computing environment, one of its goals is to support as many storage systems and file transfer protocols as possible.

Another important characteristic of Stork is reliability. It makes sure that the requested transfers are completed successfully even in case of server or network failures.

Stork source and destination URLs have a naming convention. All URLs ending with a slash (/) are assumed to be directories and the rest are assumed

to be files.

3.3 Data Transfer using Stork

3.3.1 Sample DAP submit Job File

Stork submits a submit description file which contains specification about the job to be submitted by the user. Commands that are available as part of the job description file are the following

1. dap_type = < *servicetype* > - Specify what kind of service you need from Stork. Users can specify estimation or transfer depending upon their needs

2. src_url = < *pathname* > Address of the source file that needs to be transferred by Stork. In case of estimation, users can provide host name if they need service for memory to memory data transfer between two hosts or full address path to a file if they need disk to disk estimation service

3. dest_url = < *pathname* > - Address of the destination where the files need to be transferred

4. arguments = < *argumentlist* > - List of arguments that needs to be provided to the stork server for data transfer. This is completely optional. For example, to use optimization service with history , user can provide -h

5. output = < *outputfilename* > - User who wish to see what is happening with this job can opt for this output file. Either file name or full path address of a file where users need to store their output file can be specified. Also, this can be used to capture error the program would normally write to the screen

6. error= < $errorfilename$ > - File used by stork to capture any error messages the program would write to the screen. Generally, program will show success or error depending on the transfer results.

7. x509proxy= < $pathname$ > - Used to override the default path name for X.509 user certificates. The default location for X.509 proxies is the /tmp directory, which is generally a local file system. Stork will use the proxy specified in the submit description file . Users can also specify default instead of path name.

8. checkpoint_transfer = < $Booleanvalue$ > - To have checkpointing feature enabled in Stork. If you enable here, it will override the default value in the config file

9. verify_checksum= < $booleanvalue$ > - Enables the checksum comparison of the files. If you enable here, it will override the default value in the config file

10. verify_filesize=< $booleanvalue$ > When switched on, Stork determines the file size of the files at the source and the destination and compares them.

11. file_ckpt=< $booleanvalue$ > Enable single file checkpointing.If you enable here, it will override the default value in the config file

12. disk_transfer = < $Booleanvalue$ > When mentioned in dap file while using estimation service, it will enable disk to disk estimation service

Following is a sample DAP file parameters that are possible . Users can add or modify this according to the transfer module they are using. Some dap files used for specific purposes are inside the sample dapfile folder of your installation. In case, if you install by source code, dap file folder is located in the directory where stork code resides.

```
[
dap_type="transfer or estimation";
```

```
dest_url ="gsiftp://oliver1.loni.org/work/siva/test";
src_url="gsiftp://eric1.loni.org/work/siva/18GB";
arguments = "-s 200M";
x509proxy ="default";
output ="new.out";
err ="new.err";
checkpoint_transfer   = true;
verify_filesize             = true;
file_ckpt = true
disk_transfer = "yes"


]
```

3.4 Stork Directory Structure

Stork has the following directory structure in its installation folder. They are

- bin
- etc
- log
- libexec
- sbin
- tmp

3.4.1 bin

The bin directory consists of all Stork client tools. Following are the components inside bin directory

```
[sivahpc@dsl-condor bin]$ ls
 stork_log  stork_q  stork_rm  stork_status  stork_submit
```

3.4.2 etc

etc folder consist of stork configuration files . The etc will have entries as shown below

```
[sivahpc@dsl-condor etc]$ ls
stork_config
```

Users can customize the running of stork server by editing the stork config file.

3.4.3 libexec

The libexec directory consists of all the transfer modules that you have build while installing stork.

```
stork.transfer.file-gsiftp  stork.transfer.file-petashare    stork.transfer.srm-file
stork.transfer.file-file  stork.transfer.file-http    stork.transfer.file-srb
stork.transfer.file-ftp   stork.transfer.file-irods   stork.transfer.file-srm
```

3.4.4 sbin

This directory consist stork server.

3.4.5 log

Stork server logs and history logs are located in this folder. History log will have the logs of all completed jobs. Typical entries are

```
storklog.log storkserver.log storkserver.log.history
```

3.4.6 tmp

Location in which transfer modules are executed.

3.5 Protocols Supported by Stork

Currently the following protocols and storage systems are supported by Stork:

- file
- FTP
- GridFTP
- HTTP
- iRODS
- PetaShare
- SRB

- SRM

The protocol to be used is determined by the Stork server according to the URL signatures of the files to be transferred.

3.6 URLs Supported

URLs supported: The format of the URL for various supported protocols are as below:

- file URL - file:/path/to/file
- FTP URL - ftp://user:password@host:port/path/to/file
- HTTP URL - http://user:password@host:port/path/to/file
- GridFTP URL - gsiftp://user:password@host:port/path/to/file
- SRB URL - srb://user[.mdasDomain[.zone]]:password@host:port/path/to/file
- iRODS URL - irods://user.zone:password@host:port/path/to/file
- PetaShare - petashare://path/to/file

Assuming that a service (SRB, iRODS, GridFTP, etc) is running at host on port. Please note that in the URLs shown above the parameters denote:

- user - the username
- password - password corresponding to the username
- host - the host on which the service is running
- port - the port the service listens on
- /path/to/file - path to the location of the file you would like to transfer
- /path/to/directory/ - path to the directory you would like to perform transfers

3.6.1 Sample Stork Job Requests

Sample Stork Job Requests

i) file to file transfer

```
[
  dap_type = "transfer";
  src_url = "file:/path/to/file";
  dest_url = "file:/path/to/file";
]
```

irods to file

```
[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/file";
  dest_url = "file:/path/to/file";
]
```

ii) file selection using wild cards

```
[
  dap_type = "transfer";
  src_url = "file:/path/to/file*";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "file:/path/to/*file";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "file:/path/to/fi*le";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "file:/path/to/*";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
```

```
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/file*";
  dest_url = "file:/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/*file";
  dest_url = "file:/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/fi*le";
  dest_url = "file:/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/*";
  dest_url = "file:/path/to/directory/";
]
```

iii) recursive transfer from local directory to SRB collection

```
[
  dap_type = "transfer";
  src_url = "file:/path/to/directory/";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/directory/";
  dest_url = "file:/path/to/directory/";
]
```

Any of the supported protocols may be invoked by simply replacing the URLs shown above by those of the protocols required. The new URLs should however conform to their URL format as described in the supported URL format

section above.

Chapter 4

Stork Features

4.1 Stork core features

4.1.1 File size verification support

Currently all the transfer modules supported by Stork support file size verification. File size verification can either be turned on or off by specifying the corresponding option in the Stork configuration file.

When switched on, Stork determines the file sizes of the files at the source and the files at the destination and compares them. If the file sizes differ, an error message is logged to the Stork log file.

4.1.2 Checksum verification support

Currently all the transfer modules supported by Stork support checksum verification. Checksum verification can either be turned on or off by specifying the corresponding option in the Stork configuration file.

When switched on, Stork computes the checksums of the files at the source and the files at the destination and compares them. If the checksums differ, an

error message is logged to the Stork log file.

4.1.3 Recursive transfers

Currently all the transfer modules supported by Stork support recursive directory transfers. Recursive directory transfers are specified in the URL by ending the URLs with a '/' to represent a directory.

4.1.4 Wild card Support

Currently all the transfer modules supported by Stork support transferring files with a wild cards such as *.txt, stork*, *stork or st*rk.

4.1.5 Checkpointing File Transfers

Currently the Petashare and GridFTP transfer modules supported by Stork support checkpointing of transfers and provide the capability of resuming transfers in the event of an error.

These Stork modules checkpoint the transfers during various stages and thus enable Stork to resume the transfer at the last checkpoint in the event of a network outage or crash.

4.1.6 Concurrency

Stork can handle the concurrency in the data placement jobs. Users have the option to limit the value of this concurrency. Default value is 10. Users can edit this value to their requirement in the config file.

4.1.7 Max Retry

This value represents the number of attempts stork will attempt to transfer using regular and alternative protocols. Once the number of attempts exceeds

this value, stork will return failure for the corresponding data transfer jobs. Default value is 10. Users can edit this value in the config file.

4.1.8 Run time

This feature allows the user to limit the run time for a data placement job, after which the placement is considered failure. Units are measured in minutes. Default value is 10 and minimum value should be 1. User can edit these value in the config file.

4.1.9 Stat Report

Stork submits anonymous usage statistics to the stork developers. This report contains only the version string and platform stork was built for. Default is set to true. User who does not want this feature can turn off by setting it to false in the config file.

4.1.10 Log

Stork produces both user log and server log. User log will be defined in the name storklog.log and server log will be of the name storkserver.log. Only user log will be used for any parsing of the results. Server log is meant for stork server and not for normal reading purposes.

4.1.11 Running Stork in Different Machines

Stork provides the flexibility of running stork- server in one location and stork - client in another location. This feature will be useful for many situations where a server can't run for extended period of time due to server access time limitations

Consider the following example. In this case, we are going to run `stork_server` in $dsl - condor$ machine and stork client in another machine called `dsl-stork`.

stork server is located in your sbin directory of the stork installation and clients are located in bin directory.

Step 1: Start stork server in $dsl - condor$ by running the following command

`stork_server -p port number` (Port number is the one you are going to open for the connection)

For example, `{ stork_server -p 9621}`

Step 3: Submit your jobs to the stork server from the client machine

`{stork_submit -name storkserveraddress : port number dapfile}`

For example, `{stork_submit -name dsl-condor.lsu.edu:9621 test.dap }` Stork then returns the associated job id, which is used by other Stork job control tools.

Step 4: You can do the regular stork client operation such as `{stork_status, stork_q, stork_rm }` in client's machine the same way as you did for $stork_submit$

Step 5: You can look for server log entries in server side. Also, user logs will be downloaded to client side once you have submitted the job

Stork fully supports GridFTP transfers, with the gsiftp:// protocol. To use GridFTP with Stork, users must have a valid proxy. Specify the path or say default to the created proxy using the x509proxy keyword in the Stork submit file. Placing the special value **x509proxy = " default";** or **x509proxy = " /tmp/x509up-uxxxx "**

Also, for GSIFTP transfers, you need to have client grid proxy initiated and

server grid proxy initiated. Stork will use the users credentials to authenticate to the GridFTP server.

For example,

```
[
      dap_type = "transfer";
      src_url  = "gsiftp://$src.loni.org/home/sivahpc/test/$srcfile";
      dest_url = "gsiftp://$dest.dsl-stork.org/home/sivahpc/test/dest-$destfile";
      x509proxy = "default";
      ]
```

4.1.12 Security

Security in Stork is a quiet important issue with many aspects to consider. A Stork server allows different client machines to connect to it . It is important to limit the access to this server.

Stork provides authentication by default using GSI. If no security level is needed, users can by pass the security. However, it is strongly recommended not to bypass authentication other than testing purposes .

Proper identification of a user is accomplished by the process of authentication. It is used to determine real users and malicious ones. However, many authentication mechanisms are available like Kerberos, SSL, GSI. Stork in this release supports only GSI authentication. Certificate is used to establish a trusted communication between two parties. Users needs to have their own grid credentials installed in both server and client side. They can also use their already existing grid certificates provided by various grid computing platforms such as LONI etc..

More information about the GSI (Grid Security Infrastructure) protocol which provides an avenue for Stork to do PKI-based (Public Key Infrastructure) authentication using X.509 certificates can be found in globus website. http://www.globus.org/.

Security configuration

As Stork supports only GSI authentication, it is recommended to have grid-proxies initiated at both client and server side. Stork will look for it by default. If you don't have GSI credentials, you can bypass it by following the steps below.

Anonymous authentication causes authentication to be skipped entirely. As such, it does not use authentication mechanisms, it is strongly encouraged to use for testing purposes . Users can follow the following configurations when chose to skip authentication only.

4.1.13 Client configuration

The client side needs to be configured in order to establish the authentication with the server. Users can add the following information in their config file.

```
SEC_DEFAULT_AUTHENTICATION =ANONYMOUS
SEC_CLIENT_AUTHENTICATION=ANONYMOUS
```

Server side configuration

Server side needs to be configured with the following macros in order to accept the authentication from client side. A list of acceptable methods may be provided by the daemon, using the macros

```
SEC_DEFAULT_AUTHENTICATION_METHODS
SEC_READ_AUTHENTICATION_METHODS
SEC_WRITE_AUTHENTICATION_METHODS
SEC_ADMINISTRATOR_AUTHENTICATION_METHODS
SEC_CONFIG_AUTHENTICATION_METHODS
SEC_OWNER_AUTHENTICATION_METHODS
SEC_DAEMON_AUTHENTICATION_METHODS
SEC_NEGOTIATOR_AUTHENTICATION_METHODS
SEC_ADVERTISE_MASTER_AUTHENTICATION_METHODS
SEC_ADVERTISE_STARTD_AUTHENTICATION_METHODS
SEC_ADVERTISE_SCHEDD_AUTHENTICATION_METHODS
```

```
For example, a server side may be configured (in config files) with:
SEC_DEFAULT_AUTHENTICATION_METHODS =ANONYMOUS
```

4.2 Config Files

The Stork configuration file is used to customize how Stork operates after installation. Most of the Stork installation have a basic configuration file. Any changes made to this configuration file will affect the way how Stork operates regularly.

4.2.1 Configuration file Macros

Macros definition are of the form

```
<macro_name> = <macro_definition>
```

There must be white space between the macro name, the "=" sign, and the macro definition.

Some of the Stork configuration Macros are

- RELEASEDIR = < *pathname* > This is a variable used by config file to simplify writing the path name. Default is /usr/local/stork but users can change it to their installation directory path
- PORT = < *portnumber* >Start Stork on a well-known port. Uncomment to simplify connecting to a remote Stork.
- HOST = < *ipaddressandportnumber* > Stork host may specify an optional default remote Stork server and port.
- ADDRESSFILE = < *pathname* > When Stork starts up, it can place its address (IP and port) into a file. This way, tools running on the local machine don't need an additional "-n host:port" command line option
- LOGBASE = < *pathname* > stork log base specifies the base name for heritage Stork log files
- MAXSTORKLOG = < *value* > Maximum number of entires in each log file
- MAXNUMJOBS = < *value* > Limits the number of concurrent data placements handled by Stork.
- MAXRETRY = < *value* > Limits the number of retries for a failed data placement.
- MAXDELAYINMINUTES=< *value* > Limits the run time for a data placement job, after which the placement is considered failed.
- MODULEDIR =< *pathname* > Directory containing Stork transfer modules.
- ENABLESTATREPORT = < *booleanvalue* > Enable or disable reporting of anonymous usage statistics to the Stork developers.
- RECURSIVECOPY = < *booleanvalue* > Enable or disable the recursive directory transfers
- VERIFYCHECKSUM = < *booleanvalue* > Enable or disable checksum verification support
- VERIFYFILESIZE = < *booleanvalue* > Enable or disable filesize verification support
- CHECKPOINTTRANSFER = < *booleanvalue* > Enable or disable checkpointing of file transfers

4.3 STORK LOG Features

`stork_log` is a client side tool used to retrieve a job's userlog in realtime from a remote Stork server. It can also be used to retrieve a job's userlog after the job has completed. `stork_log` is typically not invoked manually by the user, but is instead called by `stork_submit` upon submission of a job whose userlog has been requested by a "log" entry in the job submission file.

`stork_log`, upon invocation, will check for the presense of a `stork_log` daemon, and will spawn one if one is not detected. It will then send a request to the daemon to connect to the specified Stork server and transfer the userlog. This design decision allows for multiple userlog transfers to be handled by a single process.

The `stork_log` command accepts the parameters as detailed in the usage information below. `host_name` is optional and is used to specify a Stork server on a remote host. If not given, the default Stork server is assumed.

Below is the output of `stork_log --help`.

```
./stork_log [-h|--help] [-p pid] <command> [args] ...

A tool to transfer log files for Stork jobs from a Stork server to the
local machine.

Options:
  -h, --help  Print this usage information and exit
  -p <pid>    Specify the pid of the stork_log daemon to issue commands to

Valid commands and associated parameters are:
  get <host[:port]> <dap_id> <file_path>
    Retrieve a userlog for job with given dap_id from the Stork server at
    host[:port], and place it at file_path.
```

A tool to transfer log files for Stork jobs from a Stork server to the local machine. In cases, where Stork client and server is running in two different locations, stork log command can help bringing the user log to the client location. It can bring the user log per job id or users can request for the entire user log. This user log can be used for various purposes . Some output that is written on user log is shown below

```
1 000 (004.-01.000) 09/13 09:42:36 Job submitted from host: <130.39.225.155:57063>
  2 ...
  3 001 (004.-01.000) 09/13 09:42:45 Job executing on host: <130.39.225.155:54498>
  4 ...
```

```
 5 005 (004.-01.000) 09/13 09:42:45 Normal termination (return value 0)
 6 ...
 7 000 (005.-01.000) 09/13 09:48:31 Job submitted from host: <130.39.225.155:36080>
 8 ...
 9 001 (005.-01.000) 09/13 09:48:33 Job executing on host: <130.39.225.155:42576>
10 ...
11 005 (005.-01.000) 09/13 09:48:33 Normal termination (return value 0)
```

Chapter 5

Estimation and Optimization Service

Stork release 2.0 have the estimation and optimization service features. Stork 2.0 can provide the user with the optimal parallel stream number and a provision for the estimated time and throughput information for a specific data transfer.

5.1 What is Estimation and Optimization service ?

In a widely distributed many-task computing environment, data communication between participating cluster becomes a major performance bottleneck. Majority of the users fail to obtain even a fraction of the theoretical speeds promised by these networks due to issues such as sub-optimal TCP tuning, disk performance and etc. To improve the performance of the data transfer and to overcome the poor network utilization of the TCP protocol, many users uses opening up many parallel TCP streams and is highly used in many application areas.

Even these streams can enable to achieve higher throughput, opening up too many streams can cause network congestion in low-bandwidth networks and in high speed networks, it is very difficult to predict optimal parallelism level. Stork can help users in predicting optimal parallel stream number by using novel mathematical models we have developed for this purpose . A user using this service just only needs source and destination address and size of the data transfer. To the best of our knowledge, none of the existing models and tools can give as accurate results as ours with a comparable prediction overhead and we believe that Stork is unique in terms of the result it produces.

Stork can make use of this optimal parallel streams that have been predicted using estimation service for improving the throughput of the scheduled data transfers.

More information on estimation and optimization can be found in these two IEEE journal publications [1], [2]

5.2 Using Estimation Service

For users who need a first hand estimation of optimal number of parallel streams along with estimated throughput and total time for the data transfer can make use of estimation service. Stork at this time support only GridFTP protocol for its estimation and optimization service. User just needs to provide source and destination address and file size. The estimation service can make prediction by performing sampling between two sites specified in the DAP file. Based on the sampling results, Stork will do some mathematical analysis based on our model and returns result back to the user.

5.2.1 Different estimation services

Stork currently supports two types of estimation services, namely memory to memory estimation service and disk to disk data transfer estimation services. Users can just choose to follow one of the dap type depending on what type of service they require.

5.2.2 Sample Estimation service Submission DAP file

Following is an example of DAP file that can be used for the estimation service for memory to memory transfers

```
[
      dap_type = "Estimation";
      src_url  = "gsiftp://$src.loni.org/";
      dest_url = "gsiftp://$dest.dsl-stork.org/";
      x509proxy = "default";
      filesize = "2GB";
]
```

Following is an example of DAP file that can be used for the estimation service for Disk to Disk transfers

```
[
      dap_type = "Estimation";
```

[1] A Data Throughput Prediction and Optimization Service for Widely Distributed Many-Task Computing Dengpan Yin, Esma Yildirim, Sivakumar Kulasekaran, Brandon Ross and Tevfik Kosar To appear in IEEE Transactions on Parallel and Distributed Systems-Special Issue on Many-Task Computing (TPDS-SI),2011.

[2] Prediction of Optimal Parallelism Level in Wide Area Data Transfers Esma Yildirim, Dengpan Yin and Tevfik Kosar , IEEE Transactions on Parallel and Distributed Systems(TPDS),2010

```
      src_url  = "gsiftp://poseidon1.loni.org/work/siva/siva";
      dest_url = "gsiftp://eric.loni.org/work/siva/1gb";
      disk_transfer ="yes";
      x509proxy = "default";
      arguments = "-s 50M"
]
```

In the first case, file size is necessary. If not provided, stork will use 1gb as default file size. For the second case, sampling size is necessary. If not stork will use 5 percent of the file size as sampling size.

Users can submit this DAP file to Stork server just like regular stork transfer job by invoking Stork_submit command like stork_submit DAPFILE_NAME

5.2.3 Results of Estimation Service

Stork estimation service can write the results to an output file if asked in DAP file or users can just view it by using stork_status command. The following is the example output of the estimation service. Stork by default writes this output to its serverlog.history file. Users can view this result just by typing stork _ status DAPID

```
===============
status history:
===============


    [
       file_size_in bytes = 756;
        optimization_cost = 2.100000000000000E+01;
        max_throughput = 3.521927022370290E+01;
        status = "request_completed";
        dap_id = 17;
        est_time = "0 hours, 3 mins, 1 secs";
        optimal_streams = 1;
        timestamp = absTime("2010-09-22T11:09:15-0500")
    ]
```

5.3 Using Optimization Service

Optimization service in Stork provides the same functionality as like estimation service but it actually uses the estimation results to do the actual data transfer. User submit a regular data transfer job and if optimization is specified

as YES, Stork will perform the estimation service first followed by the actual data transfer using the estimation results. An another important field added to CLASSADs is use_history option. This option enforces optimization service to check from the database which keeps the history of optimized parameters from the previous transfers of specified source and destination pair. If there is such a record, stork will use the history information to perform transfers, otherwise, stork will invoke optimization service to perform estimation on the transfers and store the results into the database and then provide Stork with these results for the data transfer. Database is build in Stork by default using SQLLITE.

5.3.1 DAP sample for Optimization Service

The following are the sample DAP file for the optimization service

```
[
      dap_type = "transfer";
      src_url  = "gsiftp://$src.loni.org/";
      dest_url = "gsiftp://$dest.dsl-stork.org/";
      arguments = "-s 100M -h":

      x509proxy = "default";
      output ="tran.out";
      err= tran.err";

      ]
```

For users, who does not need history option, just remove $-h$ from the arguments field in the DAP file. Fields like err, output and log are optional for the users.

Following is the result of optimization service

```
===============
status history:
===============


    [
        status = "request_completed";
        dap_id = 10;
        timestamp = absTime("2010-09-21T12:11:43-0500")
    ]

===============
```

Chapter 6

Questions

Direct your questions to stork-devel@cct.lsu.edu, if you have any problems related to stork. To discuss any issues related to Stork, you are always welcome to use our group mailing list stork-discuss@cct.lsu.edu

Chapter 7

Appendix

7.1 Optimization and Estimation results

Figure 7.1 shows the results of optimization service that are done in 10 Gbps and 100 Mbps networks. Note that , Estimation and optimization service will predict more accurately depending on sampling size, network conditions and file size. For more information, users can refer to the following two papers

- A Data Throughput Prediction and Optimization Service for Widely Distributed Many-Task Computing, Dengpan Yin, Esma Yildirim, Sivakumar Kulasekaran, Brandon Ross and Tevfik Kosar To appear in IEEE Transactions on Parallel and Distributed Systems-Special Issue on Many-Task Computing (TPDS-SI),2011.
- Prediction of Optimal Parallelism Level in Wide Area Data Transfers, Esma Yildirim, Dengpan Yin and Tevfik Kosar , IEEE Transactions on Parallel and Distributed Systems(TPDS),2010

7.2 Stork Supported OS Platforms

You can check figure 7.3 for various Platforms/Architecture supported by Stork.

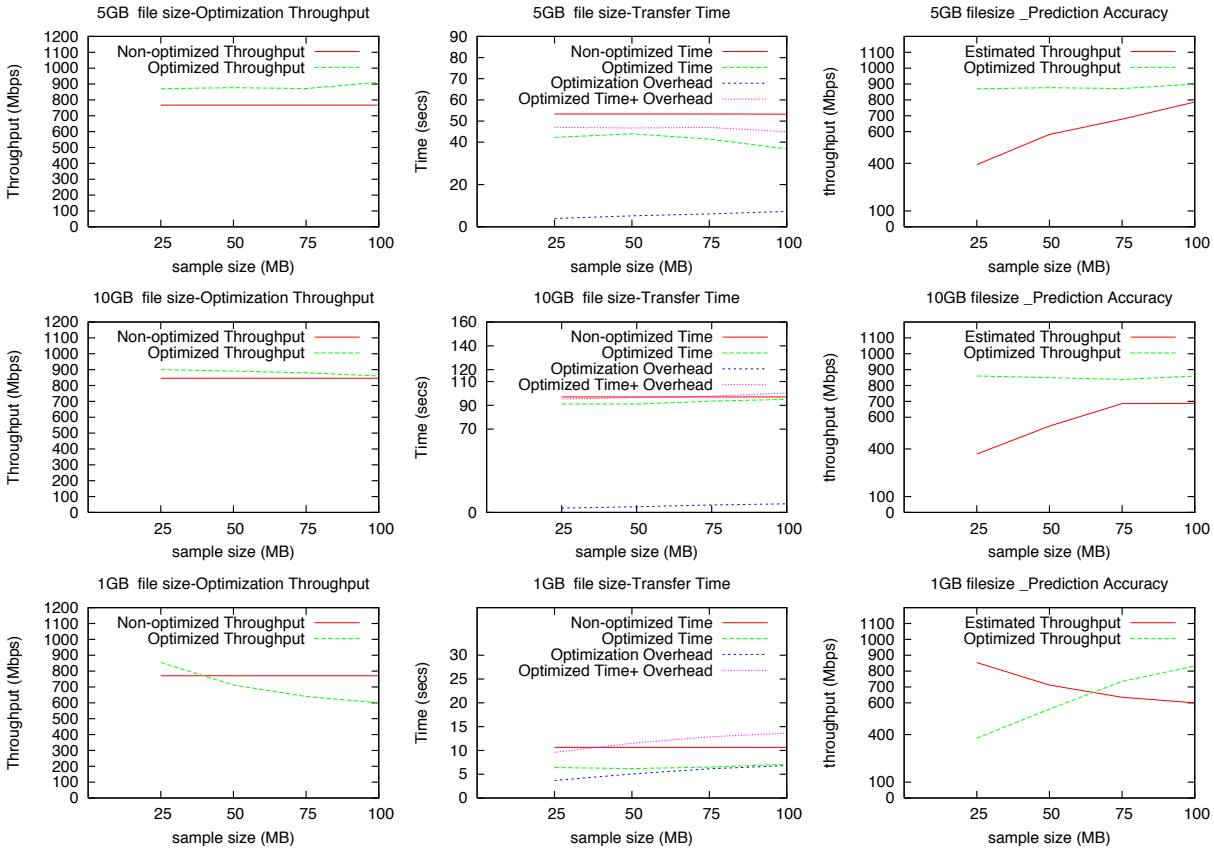

Figure 7.1: Optimization results over LONI network with 10 Gbps network interface

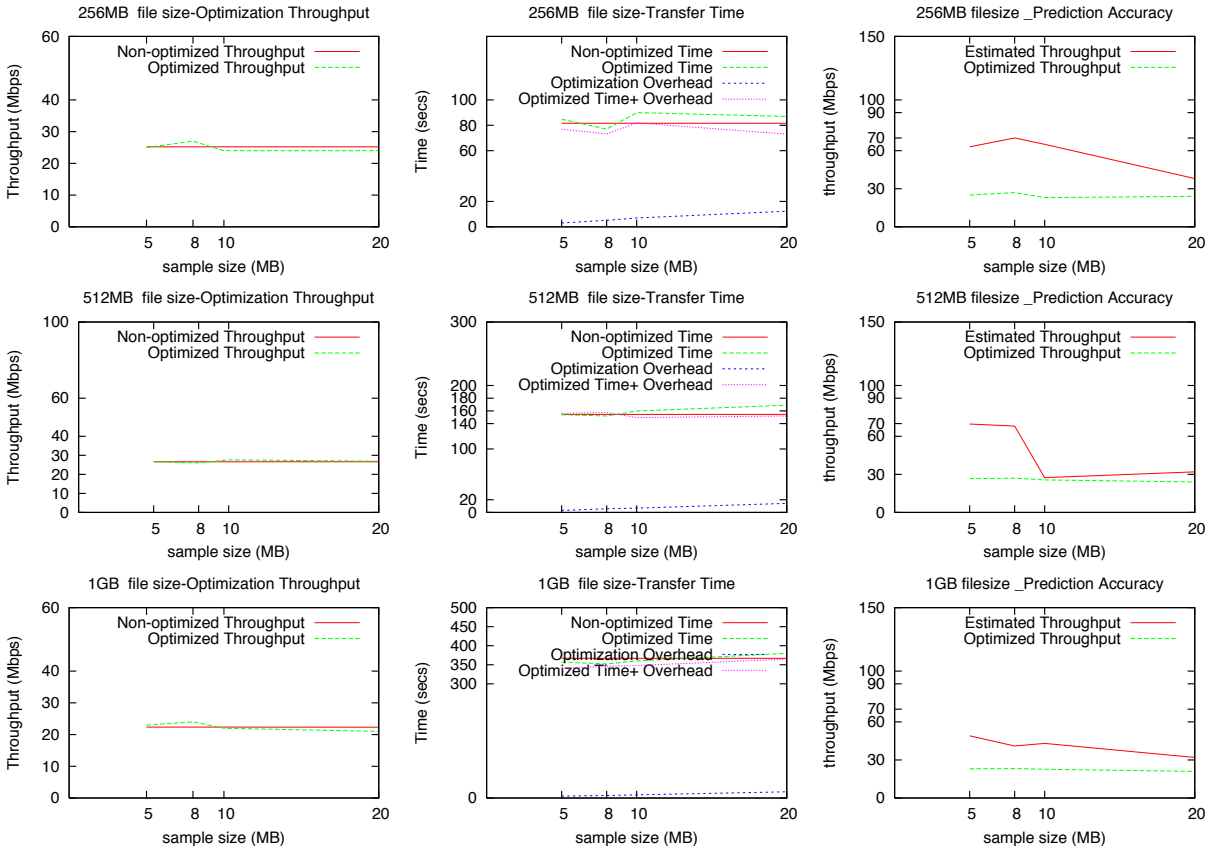

Figure 7.2: Optimization results between 100Mbps and 10 Gbps LONI network Interface

Platforms/Architectures	Core Stork	Globus	SRB	iRODS	Petashare	SRM
x86_rhap_5	Yes	Yes	Yes	No	No	Yes
x86_64_rhas3	Yes	Yes	Yes	Yes	Yes	Yes
x86_64_rhap_5.3 updated	Yes	Yes	Yes	Yes	Yes	Yes
ia64_rhas_3	Yes	Yes	Yes	No	No	Yes
x86_deb_4.0	Yes	Yes	Yes	No	No	Yes
x86_64_deb_5.0	Yes	Yes	Yes	Yes	Yes	Yes
x86_rhas_3	Yes	Yes	Yes	No	No	Yes
x86_64_rhap_5.2	Yes	Yes	Yes	Yes	Yes	Yes
x86_64_rhap_5	Yes	Yes	Yes	Yes	Yes	Yes
x86_64_rhap_5.3	Yes	Yes	Yes	Yes	Yes	Yes
x86_64_rhas_4	Yes	Yes	Yes	Yes	Yes	Yes
x86_sles_9	Yes	Yes	Yes	No	No	Yes
x86_64_sles_9	Yes	Yes	Yes	Yes	Yes	Yes
x86_deb_5.0	Yes	Yes	Yes	No	No	Yes
x86_rhas_4	Yes	Yes	Yes	No	No	Yes
X86_64_Ubuntu_10.04	Yes	No	No	No	No	No
ps3_Fedora_9	Yes	No	No	No	No	No
ps3_ydl_5.0	Yes	No	No	No	No	No
x86_64_ubuntu_10.04	Yes	No	No	No	No	No
x86_64_fedora_11	Yes	No	No	No	No	No

Figure 7.3: Stork Supported Platform and Architectures