Performance Testing Framework Project Plan

Draft

Kent Yang, Quincey Koziol, Arash Termehchy

I. Purposes

1. Speed up the HDF5 library debugging processes
2. Enhance the confidence for HDF developers
3. Provide quick feedbacks to HDF HelpDesk
4. Provide concrete and easy-to-use performance testing suite for HDF5 applications
5. Provide good presentation materials for outreach activities

II. Requirements

1. Framework controls the front-end(input) and back-end(output).
2. A script file controls all the individual tests.
3. **Each test should be kept as a standalone program if possible.**
4. An internal library may be necessary to efficiently pass information from standalone program back to the framework for further analysis.
5. Easy to add/delete individual components.
6. Output should support the followings:
    a. An option to generate a standard ASCII text file that can be read by standard editors such as “vi” under UNIX environment.
    b. It should also be passed to the back-end database.
7. The output should be sent to a server to publish the results; Individual user can set an option not to go to this step.
8. The output should be analyzed statistically. Database should be used for this purpose.
9. Database should be open-source; MySQL is an option.
10. Gnu plot may be an option to generate final plots for the output. Two kinds of reports should be generated: a summary report and a detailed report.
11. Final outputs should be posted at the web site.

III. Language

C++
Why?

- A good object-oriented language
- Easy to be picked up by C programmers
- Easy to add/delete/modify tests
- Stable on most platforms

IV. Remaining issues

1. Input

We had some discussions on whether to use configuration files or command-line options. Both methods have pros and cons.

The third option is to use the server database as the input. This option will benefit for our own performance testing programs.

Since we have already had standalone tests such as h5perf with command-line options; performance tests may be very diverse and it will be difficult to describe these diversities with common configuration files; the consensus is to use command-line options at first.

However, our framework should be designed in such a way so that it can be easy to add the usage of the configuration files or the database in the future.

2. Anything else?

V. Implementation details

1. James will help to set up auto-configuration/make file.
2. Arash will implement the framework.
3. Steps:
   1) Set up the framework that can generate text output.
   2) Enhance the performance library so that the output can also be stored in the database.
   3) Set up a separate server to manage the database.
   4) Connect one client to the server.
   5) Connect multiple clients to the server.
   6) Tune in the framework.
   7) Add the Chicago performance test into the framework.
4. Suggestions for Functions of Performance library:
   1) Function calls for general input options for all tests
   2) Function calls to handle information stored at the database
      (System configuration, time, etc.)

VI. Flow chart

Some illustrations of the chart

1. Script file:
is an executable file that uses the script language to run all testing programs with various options. The following is a simple example; it is just for the purpose of illustration.

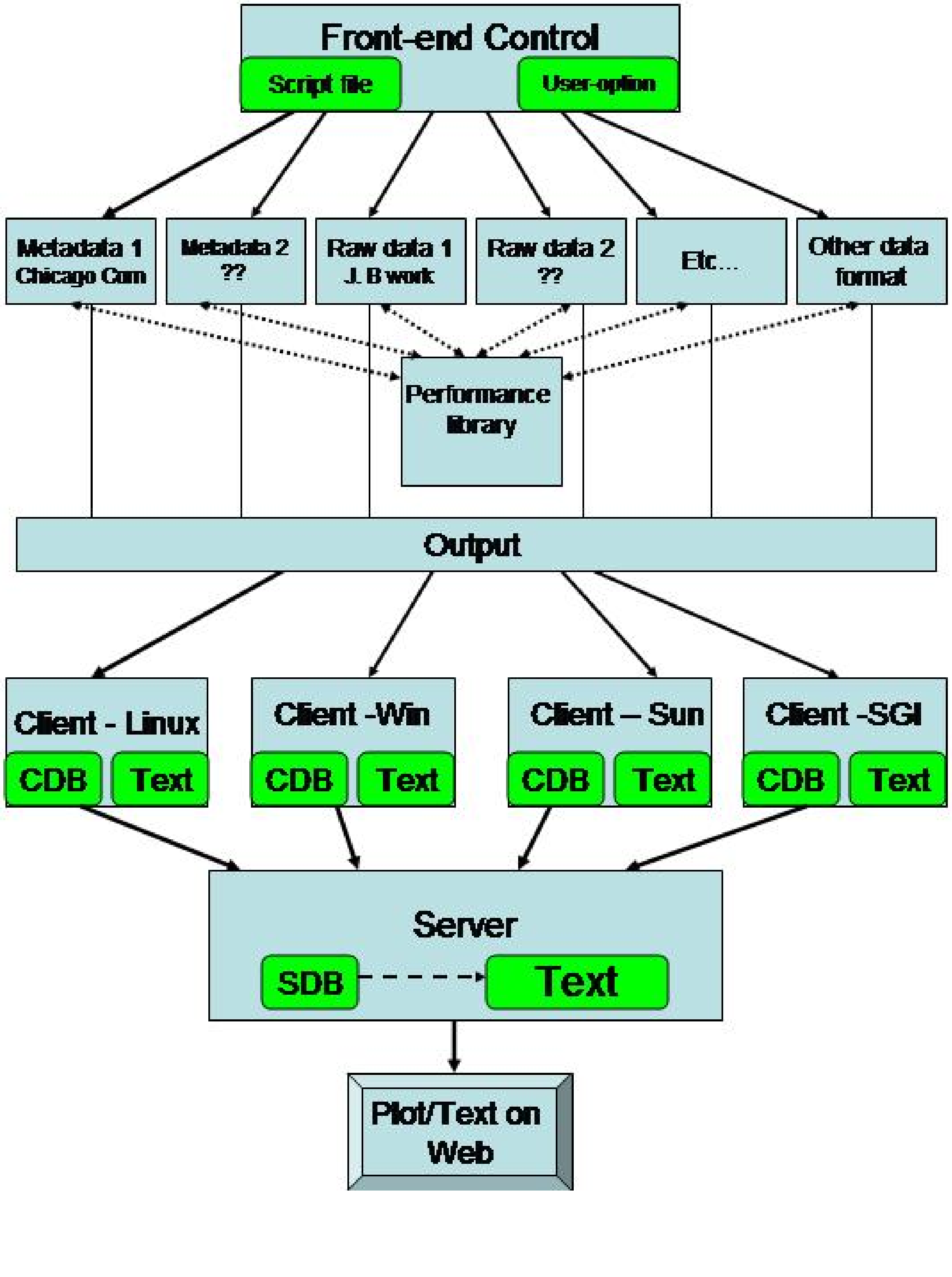
```
Mdperf –nd 2000000 –dim 1000000
Mdperf –ng 20000
….
```

2. User-option:
The users can also use “user-option” to run their own performance test.

```
Mdperf –ng 100
```

3. CDB: Client Data Base. This can be reduced or avoided if we can make the server obtain the output directly.
4. SDB: Server Data Base. It can be used to generate the final output.

Performance Testing Frame work



Front-end Control
Script file
User-option
Metadata 1 Chicago Com
Metadata 2 ??
Raw data 1 J. B work
Raw data 2 ??
Etc…
Other data format
Performance library
Output
Client - Linux
Client -Win
Client – Sun
Client -SGI
CDB
Text
Server
SDB
Text
Plot/Text on Web