BRL-CAD: Benchmark Performance Database

August 20, 2012

Contents

1 Database

1.1 Idea

The main idea is to store the data both in the database and the log files stored in the archive. The storage in the database enables the content to be searched via the parameters such as machine descriptions, versions, results and could be compared. The storage as a file in the archive is to maintain a back up of the file and whenever a developer needs a specific bunch of benchmark logs, he could get them from the archive with or without the help of the database.

1.2 Schema

```
--
-- Table structure for table ‘benchmark_logs‘
--

DROP TABLE IF EXISTS ‘benchmark_logs‘;
CREATE TABLE IF NOT EXISTS ‘benchmark_logs‘ (
  ‘id‘ int(11) NOT NULL AUTO_INCREMENT,
  ‘machine_desc‘ varchar(255) DEFAULT NULL,
  ‘brlcad_version‘ varchar(15) DEFAULT NULL,
  ‘running_time‘ int(10) DEFAULT NULL,
  ‘time_of_execution‘ timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_
```

```
  `approx_vgr` float(10,5) DEFAULT NULL,
  `log_vgr` float(10,5) DEFAULT NULL,
  `params` varchar(255) DEFAULT NULL,
  `results` varchar(255) DEFAULT NULL,
  `compiler_flags` text,
  `complete_info` text,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `email_logs`
--

DROP TABLE IF EXISTS `email_logs`;
CREATE TABLE IF NOT EXISTS `email_logs` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `imap_id` int(10) DEFAULT NULL UNIQUE,
  `md5` varchar(100) DEFAULT NULL,
  `time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `id` (`id`),
  KEY `md5` (`md5`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `email_verification_logs`
--

DROP TABLE IF EXISTS `email_verification_logs`;
CREATE TABLE IF NOT EXISTS `email_verification_logs` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `verified_till` int(10) DEFAULT NULL,
  `time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `verified_till` (`verified_till`),
  KEY `time` (`time`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `machine_info`
--

DROP TABLE IF EXISTS `machine_info`;
CREATE TABLE IF NOT EXISTS `machine_info` (
  `benchmark_id` int(11) DEFAULT NULL,
  `osrelease` varchar(100) DEFAULT NULL,
```

```
  `hostname` varchar(100) DEFAULT NULL,
  `cores` int(10) DEFAULT NULL,
  `processors` int(10) DEFAULT NULL,
  `physical_addr_size` int(5) DEFAULT NULL,
  `virtual_addr_size` int(5) DEFAULT NULL,
  `vendor_id` varchar(100) DEFAULT NULL,
  `ostype` varchar(100) DEFAULT NULL,
  `cpu_mhz` float(10,5) DEFAULT NULL,
  `model_name` varchar(150) DEFAULT NULL,
  KEY `hostname` (`hostname`),
  KEY `cores` (`cores`),
  KEY `ostype` (`ostype`),
  KEY `processors` (`processors`),
  KEY `vendor_id` (`vendor_id`),
  KEY `benchmark_id` (`benchmark_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ------------------------------------------------------

--
-- Table structure for table `md5_log`
--

DROP TABLE IF EXISTS `md5_log`;
CREATE TABLE IF NOT EXISTS `md5_log` (
  `benchmark_id` int(11) DEFAULT NULL,
  `file_name` varchar(255) DEFAULT NULL,
  `md5sum` varchar(100) DEFAULT NULL,
  `archived` enum('YES','NO') DEFAULT NULL,
  `db_entries` enum('YES','NO') DEFAULT NULL,
  KEY `md5sum` (`md5sum`),
  KEY `benchmark_id` (`benchmark_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ------------------------------------------------------

--
-- Table structure for table `rt_average`
--

DROP TABLE IF EXISTS `rt_average`;
CREATE TABLE IF NOT EXISTS `rt_average` (
  `benchmark_id` int(11) DEFAULT NULL,
  `abs_rps` float(20,5) DEFAULT NULL,
  `vgr_rps` float(20,5) DEFAULT NULL,
  `result` enum('RIGHT','WRONG','COMPARISION_FAILURE') DEFAULT NULL,
  KEY `benchmark_id` (`benchmark_id`),
  KEY `result` (`result`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ------------------------------------------------------
```

```
--
-- Table structure for table `rt_bldg391`
--

DROP TABLE IF EXISTS `rt_bldg391`;
CREATE TABLE IF NOT EXISTS `rt_bldg391` (
  `benchmark_id` int(11) DEFAULT NULL,
  `abs_rps` float(20,5) DEFAULT NULL,
  `vgr_rps` float(20,5) DEFAULT NULL,
  `result` enum('RIGHT','WRONG','COMPARISION_FAILURE') DEFAULT NULL,
  KEY `benchmark_id` (`benchmark_id`),
  KEY `result` (`result`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `rt_m35`
--

DROP TABLE IF EXISTS `rt_m35`;
CREATE TABLE IF NOT EXISTS `rt_m35` (
  `benchmark_id` int(11) DEFAULT NULL,
  `abs_rps` float(20,5) DEFAULT NULL,
  `vgr_rps` float(20,5) DEFAULT NULL,
  `result` enum('RIGHT','WRONG','COMPARISION_FAILURE') DEFAULT NULL,
  KEY `benchmark_id` (`benchmark_id`),
  KEY `result` (`result`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `rt_moss`
--

DROP TABLE IF EXISTS `rt_moss`;
CREATE TABLE IF NOT EXISTS `rt_moss` (
  `benchmark_id` int(11) DEFAULT NULL,
  `abs_rps` float(20,5) DEFAULT NULL,
  `vgr_rps` float(20,5) DEFAULT NULL,
  `result` enum('RIGHT','WRONG','COMPARISION_FAILURE') DEFAULT NULL,
  KEY `benchmark_id` (`benchmark_id`),
  KEY `result` (`result`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `rt_sphflake`
--
```

```
DROP TABLE IF EXISTS `rt_sphflake`;
CREATE TABLE IF NOT EXISTS `rt_sphflake` (
  `benchmark_id` int(11) DEFAULT NULL,
  `abs_rps` float(20,5) DEFAULT NULL,
  `vgr_rps` float(20,5) DEFAULT NULL,
  `result` enum('RIGHT','WRONG','COMPARISION_FAILURE') DEFAULT NULL,
  KEY `benchmark_id` (`benchmark_id`),
  KEY `result` (`result`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------------------------------------

--
-- Table structure for table `rt_star`
--

DROP TABLE IF EXISTS `rt_star`;
CREATE TABLE IF NOT EXISTS `rt_star` (
  `benchmark_id` int(11) DEFAULT NULL,
  `abs_rps` float DEFAULT NULL,
  `vgr_rps` float DEFAULT NULL,
  `result` enum('RIGHT','WRONG','COMPARISION_FAILURE') DEFAULT NULL,
  KEY `benchmark_id` (`benchmark_id`),
  KEY `result` (`result`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------------------------------------

--
-- Table structure for table `rt_world`
--

DROP TABLE IF EXISTS `rt_world`;
CREATE TABLE IF NOT EXISTS `rt_world` (
  `benchmark_id` int(11) DEFAULT NULL,
  `abs_rps` float DEFAULT NULL,
  `vgr_rps` float DEFAULT NULL,
  `result` enum('RIGHT','WRONG','COMPARISION_FAILURE') DEFAULT NULL,
  KEY `benchmark_id` (`benchmark_id`),
  KEY `result` (`result`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

1.3 Implementation

The database chosen for this project is MySQL.

2 Code Structure

The following is the layout followed for the project.

```
trunk/
    htdocs/
        benchmark/
            archive/
            BenchmarkExtension/
                BenchmarkExtension.php
                BenchmarkExtension.api.php
                SpecialBenchmarkExtension.php
            config
            doc/
            __init__.py
            libs/
                bp_logger.py
                charting.py
                dbutils.py
                imaputils.py
                __init__.py
                parser.py
                util.py
            log/
                benchmark.log
            scripts/
                process_from_queue.py
                parse.py
                get_email_attachments.py
                email_verifier.py
                queue_verifier.py
                benchmark_upload.py
            src/
                __init__.py
                controllers/
                    api.py
                    imgperf_vs_cpus.py
                    perf_vs_arch.py
                    perf_vs_cpus.py
                    perf_vs_images.py
                    upload.py
                views/
                    index.tpl
                    plot.tpl
            sql/
                benchmark_db.sql
```

The libs folder has the core libraries required for parsing, utilities required for database usage, IMAP server and some generic utils and a custom logger which is a wrapper over the logging module of python. The logging level has been set to logging.ALL so that each detail could be added to the logfile for a fruitful and easy debugging.

The scripts folder currently consists of the scripts to parse the file passed as an argument, a script which could be scheduled to parse the files in the queue folder, a script which gets all the unread emails and verifiers to check if the data transfer has taken place well or not.

BenchmarkExtension folder contains the mediawiki extension setup, the implementation of the custom API action and the special page which could be used to access the

data in the database.

Config file provides the various configuration options for the project. These settings are used whereever needed. An example configuration file is as follows.

```
[database]
host = localhost
username = root
password = <password>
database = benchmark_db

[locations]
archives = ./archive
log = ./log
queue = /srv/http/queue

[imap_credentials]
imap_server = imap.gmail.com
imap_port = 993
username = benchmark.performance.database
password = <password>
```

All the above mentioned fields are needed.

3 File naming convention

The file name of the log file is in the format "run-<process number>-benchmark.log". The <process number> could the same on multiple machines and different log files submitted could have the same name. To take care of this problem, the file in the queue folder and the archive is renamed to the following format, "run-<process number>-benchmark.log.<md5 hex digest of the content of the log>".

4 Parser

4.1 Idea

The data of the log file needs to be parsed and the relevant and important information has to be logged in the database so as to make the data access for the graphic display through the frontend could be made possible.

4.2 Implementation

The parser has been implemented in a modular fashion. The advantage of the modular design of the parser is that one could implement more "data extractor" methods with out much effort.

Every required field in the log file is extracted using a regular expression. These regular expressions are based on the text in the brlcad/bench/run.sh file. Any changes to the text there would affect the extraction of the data via these regex patterns.

The main information that is being captured here contains BRL-CAD version, machine description, running time, time of execution, approximate vgr, logarithmic vgr, params and results. This data is written into the benchmark_logs table of the database. The complete log file is also stored as text in the database and this field could be used for full-text search.

The BRL-CAD version is reported by different libraries in the system. The parser checks if all the libraries report the same version number. If not, report that that there is a mismatch.

The machine description which is obtained from uname is stored into the db. The performance metrics are captured as per the regexes.

The results of the tests and params are stored in a dictionary/json format.

```
{'star': 'RIGHT', 'm35': 'RIGHT', 'bldg391': 'RIGHT',
'moss': 'RIGHT', 'world': 'RIGHT', 'sphflake': 'RIGHT'}
```

Along with this, the cpuinfo and kernel parameters are also parsed using regex patterns and logged into the machine description table. Once the frontend is created, one could actually check and thus compare the benchmark log statistics using the cpuinfo and kernel parameters.

5 Web API

5.1 Idea

Http API is a webservice that runs to which the user could submit his benchmark data, this may be independent(preferably) to the existing web services of brl-cad and ideally they could interact via the db.

5.2 Implementation

This is implemented as an extention of mediawiki. This is named as BenchmarkExtension. The setup file is BenchmarkExtension.php which declares all the necessary stuff required to be loaded by mediawiki to execute the plugin. A custom API action *benchmark* has been implemented which is different from the internal upload API action. I've tried to use the native uploadBase code to implement this API but I couldn't do that. Thus this was implemented as a direct full file upload via POST. The parameters for the API call are as follows.

```
'action' => 'benchmark',
'filename' => 'the name of the file',
'content' => 'File contents',
'comment' => 'same as the description',
```

The file is then transferred to the queue folder.

6 IMAP

6.1 Idea

The benchmark log files are mailed as attachments to benchmark@brlcad.org at the moment. To gather these log files automatically, a script was needed. The main duty of the script is to move the attachments to the queue folder.

6.2 Implementation

This particular module uses the imaplib library of python to gain access to the unread emails. For each of the unread emails, the attachments are checked to see if a log file with the necessary regex pattern for the filename is found. If such a file is found, it

is moved to the queue folder with the necessary changes to the filename as mentioned above.

Sometimes the process might break in between. To ensure that all the attachments of the read mails are properly backed up and the data being in the archive or the queue folder, we could use the email_verifier script.

7 Frontend

7.1 Idea

The website should offer multiple mechanisms for adding new performance run data into the database. The website should provide multiple graphical and non-graphical visualizations of aggregate performance data (i.e., graphs, charts, tables, etc).

7.2 Implementation

The frontend is implemented as an MVC architecture using the python microframework *bottle*. For the charting a library has been written which acts as a wrapper over Google Charts. A controller has been implemented for each of the plots. This controller handles the forms and the rendering of the graph or the table as asked. This interacts with the database and renders the form and the graphs. The controller uses the charting library and renders the data via the template in the views folder.

When the page is loaded, it is a `GET` request and just loads the form. When the form is submitted, the data is submitted as a `POST` request and this instantiates controller methods which build the appropriate representation of the data or send out on error message through the view.

7.3 Charting Library

Charting library has been written as a wrapper to access the Google Charts. There was a GCharts image API but that has been deprecated to bring in the new API which could be accessed by embedding javascript code into the webpage where the plot has to be displayed.

The data is sent to the library as a dictionary with the additional information of which forms the base axis. The library re-arrages the data and builds the javascript code using the data. This code is then embedded into the webpage.

7.4 Plot View

The template for the plot consists of three placeholders, for javascript code, html form and error messages. When the page is loaded as `GET` request, only the form is loaded and the other two placeholders are substituted by an empty string and when the page is loaded as `POST` request, the javascript code obtained via the controller methods is used to substitute the javascript code placeholder. All the errors are spewed out via the error messages placeholder.

7.5 Available Plots

- **Absolute Performance vs Reference images** : Plot of absolute performance(rays/sec) of various architectures w.r.t to each of the reference images. The horizontal axis has the reference images and the vertical axis is the average of all the measured absoluteperformances over a particular architecture. One could compare various architectures w.r.t to these images.

- **Average Absolute Performance vs Architecture** : Check the variation of average of the absolute performance(rays/sec) on various architectures. The horizontal axis has the architectures and the vertical axis measures the average of absolute performance. Atleast two architectures have to be selected for this plot.
- **Performance vs Number of CPUs** : Performance variation with the number of CPUs over various architecture with the variation could be checked using this plot. The horizontal axis has the number of CPUs and the vertical axis measures the average of absolute performance.
- **Image Performance vs Number of CPUs** : Performance variation with the number of cores over various reference images over a particular architecture. The horizontal axis has the number of CPUs and the vertical axis measures the average of absolute performance.