International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-1 Issue-6, August 2012

# Job Execution Framework for Performance Testing

**Mangala S Joshi, K Nirmala Kumari, Saumil G Merchant**

***Abstract*— Performance Testing [2] determines the system behavior under specific system workloads. The process involves identifying the attributes and acceptance criteria followed by design of tests and configuration of test environment. These tests are implemented and executed on the system. Validation of tests, results collection and analysis of the same is carried out in the next step. Implementing and executing a number of tests either at the instant or schedule it to execute at a particular time on the system would be easier and helpful as it saves tester's effort and time. Job Execution Framework (JEF) is generic in nature and fulfills the above requirement. It supports 10 functions of which 8 are open to users. This paper gives detailed description of all the functions and their usage.***

## I. INTRODUCTION

In general, Performance testing[1] is in general, verification performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also be used to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage. Performance testing is a subset of performance engineering, which strives to build performance into the design and architecture of a system, prior to the onset of actual coding effort. The purpose of performance testing is to demonstrate that the system meets performance criteria, compare two system meets performance criteria and measure what parts of the system or workload causes the system to perform badly.

Major types of performance testing[1] are: Load Testing-A simple test conducted to understand the behavior of the system under a specific expected load. This load can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration. This test will give out the response times of all the important transactions. Stress testing: Stress testing is normally used to understand the upper limits of capacity within the system. This kind of test is done to determine the system's robustness in terms of extreme load. It enables administrators to make sure if the system performs adequately if the current load goes well above the expected maximum. Endurance Testing: It determines if the system is able to withstand the continuous expected load. During endurance tests, memory utilization is monitored.. It involves applying a significant load to a system for an extended, significant period of time. Spike Testing: It determines if performance deteriorates, system fails, or it can handle drastic changes in load. Spike testing is done by suddenly increasing load generated by users by a large amount and observing the behavior of the system. Configuration Testing: It determines the effects of configuration changes to the system's components on the system's performance and behavior.

Performance Testing Methodology [4] involves basically five steps as shown in the diagram.

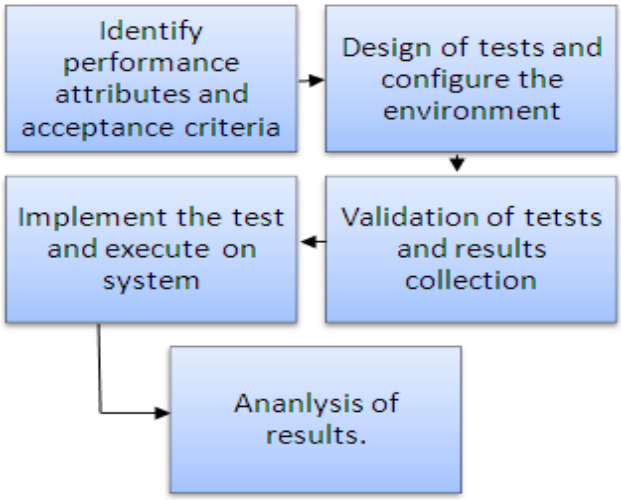



Fig .1: Block Diagram of performance testing methodology.

Identify the response time, throughput, and resource utilization goals and constraints. Additionally, identify project success criteria that may not be captured by those goals and constraints; for example, using performance tests to evaluate what combination of configuration settings will result in the most desirable performance characteristics. . Identify key scenarios, determine variability among representative users and how to simulate that variability, define test data, and establish metrics to be collected. Consolidate this information into one or more models of system usage to be implemented, executed, and analyzed. Prepare the test environment, tools, and resources necessary to execute each strategy as features and components become available for test. Ensure that the test environment is instrumented for resource monitoring as necessary. . Develop the performance tests in accordance with the test design. Run and monitor your tests. Validate the tests, test data, and results collection. Execute validated tests for analysis while monitoring the test and the test environment. Analyze, Consolidate and share results data. Make a tuning change and retest. Each improvement made will return smaller improvement than the previous improvement.

**Manuscript published on 30 August 2012.**
* Correspondence Author (s)
**Mangala S Joshi*,** M.Tech, VLSI Design &Embedded Systems , Dept of E & C, BIT, Bangalore
**K Nirmala Kumari,** Asso.Prof, Dept of E & C, BIT, Bangalore.
**Saumil G Merchant,** [3]Systems and Technology Group, IBM India Pvt Ltd.

## II. ARCHITECTURE OF JOB EXECUTION FRAMEWORK (JEF)

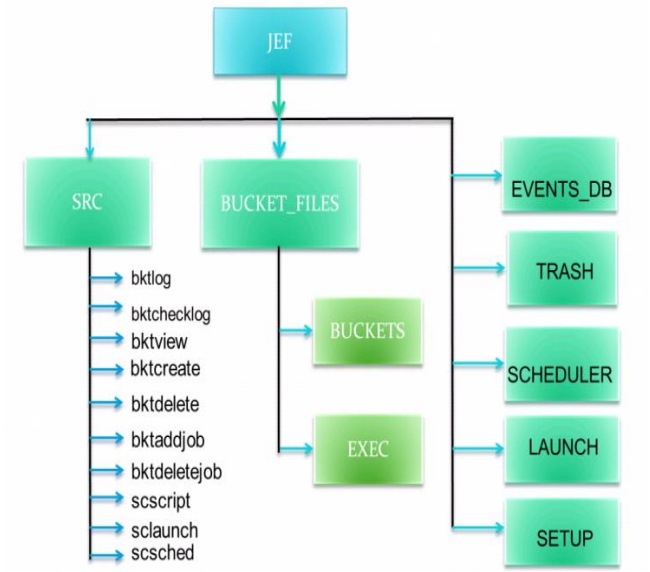



Figure: 2 JEF Architecture

In the above process, implementing and executing a number of tests either at the instant or schedule it to execute at a particular time on the system would be easier and helpful as it saves tester's effort and time. Job Execution Framework(JEF) is a generic tool which fulfils the above mentioned criteria and finds use in places which require to automate creation and submission of jobs as a collection called "Buckets" and serves to execute it either directly or time schedule it using Cron[3]. This tool is implemented in Perl completely and runs on Linux platform.

The specification of the framework is to automate the creation of bucket file and add jobs into a given bucket. Deletion of jobs from buckets and deletion of entire bucket files.The architecture of JEF is shown above.It has basically 7 directories. SRC holds the source scripts of all the functions it support and a directory "LOG" to hold the log files of the source scripts. These log files are collected with a timestamp as it would help debug in any undesired behavior. BUCKET_FILES has 2 sub-directories i,e BUCKETS and EXEC. BUCKETS directory holds all the bucket files with an extension ".bkt". EXEC has all the exec scripts each of which is an executable form of a bucket file or multiple files together.


Figure: 3 JEF Directory

In order to keep track the status of existing buckets, a database file is maintained. This file "bucketDB" is located in EVENTS_DB directory. User can view this database file using function "bktchecklog". This file logs the bucket ID and creation time when the bucket is created. When jobs are added or deleted to/from bucket, this file gets updated automatically and reflects the existing number of jobs and updating time. Hence, user can just check this database file anytime to know about all the existing buckets and their contents. If the user wants to check the content of any bucket, he may do so by using the function "bktview" which displays the entire contents of the bucket.

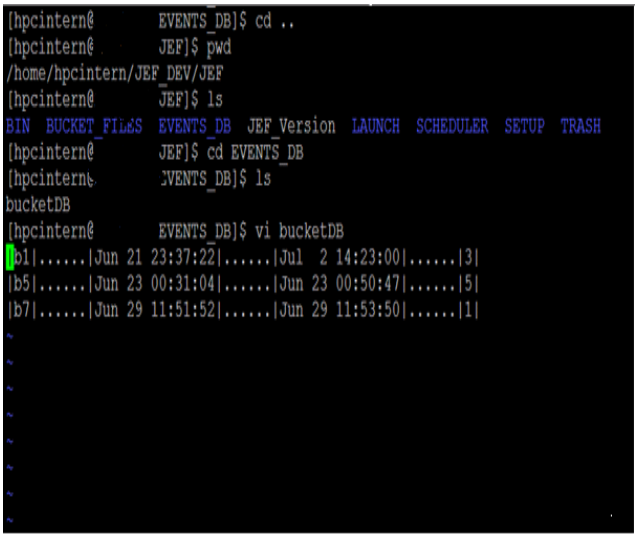



Figure: 4 JEF Database File bucketDB

TRASH holds the deleted bucket files in case it would be required later for use. JEF supports 10 functions of which 8 are open to users and are listed below.

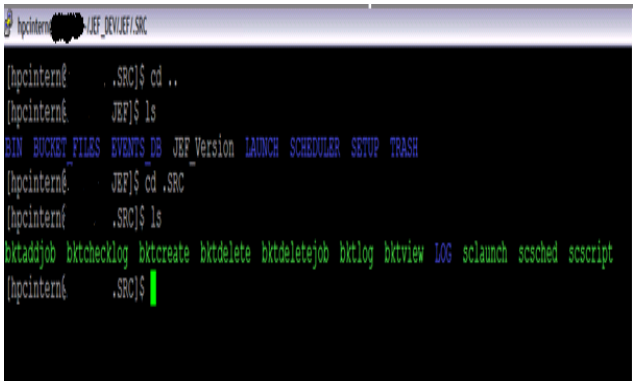



Figure: 5 JEF Database File bucketDB

## III. JEF FUNCTIONS

1. **Bktlog** This is not open to users rather used internally in scripts for automatically updating the database file bucketDB. This basically has 3 functions -

i) **add** This option along with bucket ID logs it into the database file bucketDB when the bucket is created. Hence, number of jobs will be 0 and updating time is same as creation time.
**Eg: ./bktlog –buckId b1 -add**

ii) **Delete** This deletes the specified bucket file entry from bucketDB.
**Eg: ./bktlog –buckId -delete**

iii) **update**This updates the specified bucket file entry with the given number of jobs and also updates the time.
**Eg: ./bktlog –buckId b1 –num_of_jobs 5**

2. **bktchecklog** This is used to check the database file in two modes. It checks if a bucket with the specified Id exists or not by returning a value to the calling script and in bverbose mode, it prints out the complete status of that bucket.


Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

**Eg: ./bktchecklog –buckId b1,**
**./bktchecklog –b buckId b1 –v**

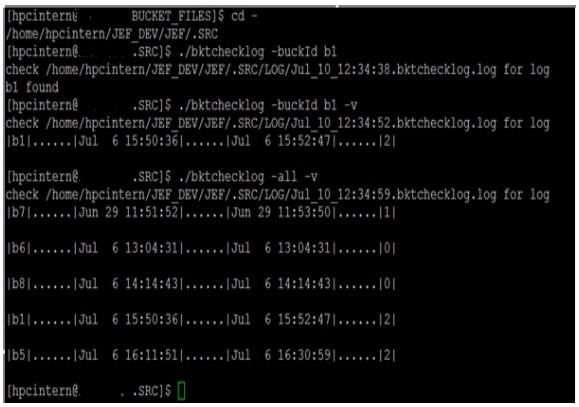



Figure: 6 JEF bucketchecklog

3. **bktcreate** It creates an empty bucket in the directory $JEF_PATH/BUCKET_FILES/BUCKETS with the name buckId.bkt.

**Eg:./bktcreate –buckId b1**

This would create the bucket with name b1.bkt at the mentioned location. It also checks if a bucket with the input ID already exists. If found, it errors out with a message. And it gets logged into the bucketDB file automatically.

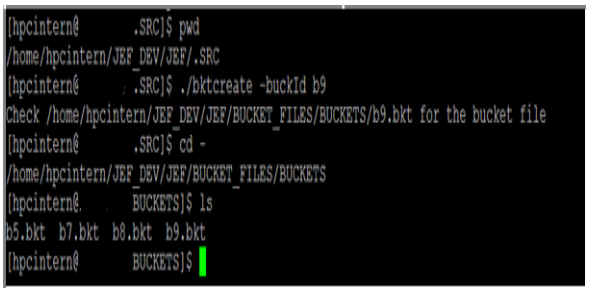



Figure: 7 JEF Bucket create

4. **Bktdelete** This deletes the specified bucket from $JEF_ ATH/BUCKET_FILES/ BUCKETS directory and saves a copy in TRASH directory.

**Eg: ./bktdelete –buckId b1**

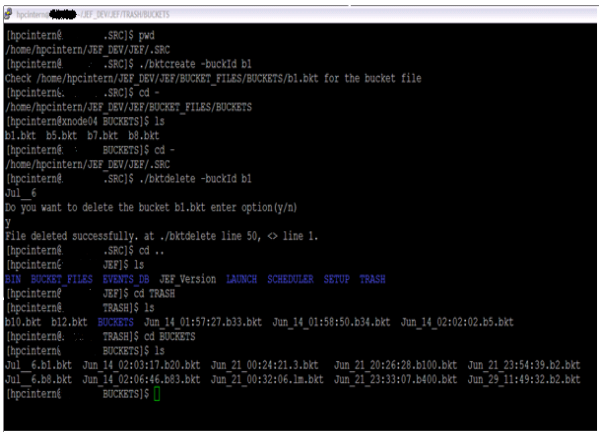
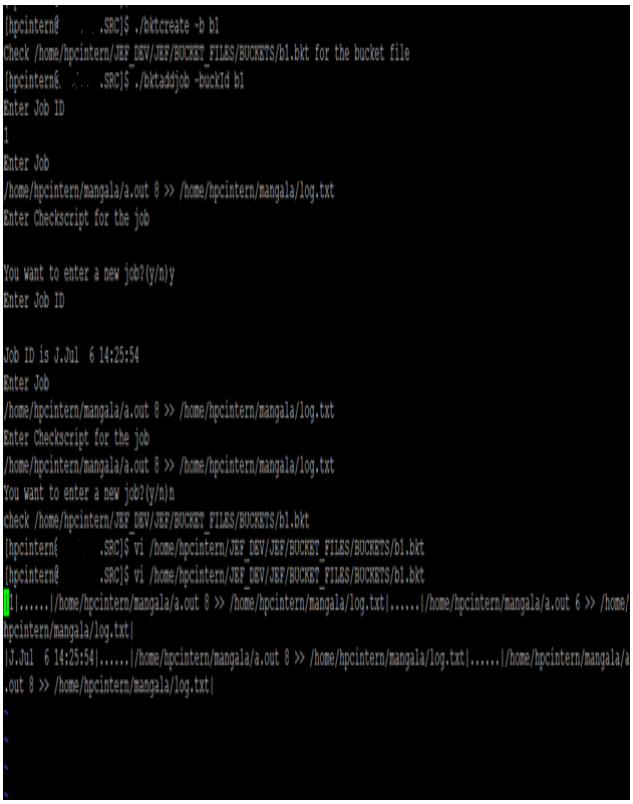



Figure: 8 JEF Bucket Delete

5. **bktaddjob** It adds jobs to the specified bucket file in two modes-

1) Interactive: In this mode, num_of_jobs option is not specified hence the script keeps asking the user if he wants to continue adding job or not. Depending on the input, it adds or exits.

**Eg: ./bktaddjob –buckId b1**



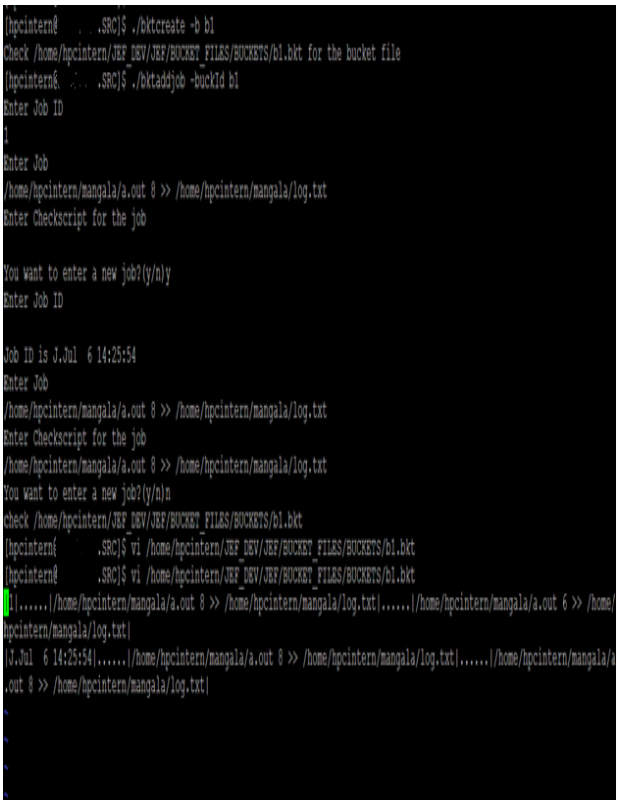




Figure:9 Adding jobs to bucket in interactive mode


Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

2) Non-interactive: In this, num_of_jobs option is specified hence, those many jobs are added one after the other and it exits.

**Eg: ./bktaddjob –b buckId –num_of_jobs 5**

It requires user to enter unique jobId and job followed by check script which is optional. If the user does not enter jobId, the script creates one. The bucketDB file also gets updated automatically.

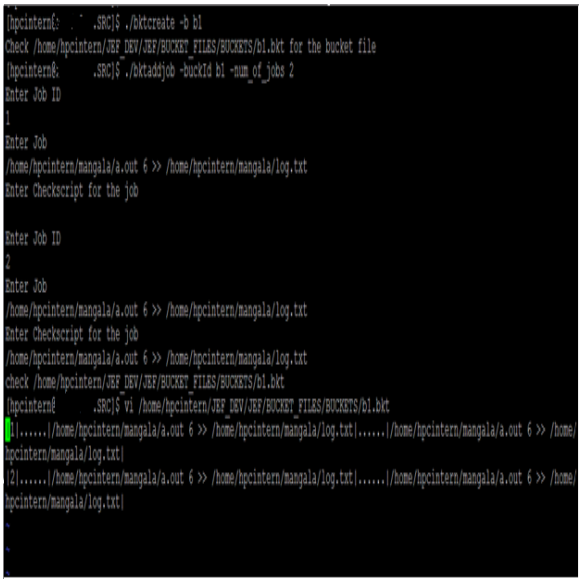


Figure: 10 Adding jobs to bucket in non-interactive mode

6. **Bktdeletejob** It deletes jobs with the id specified for the bucket. This also supports interactive and non-interactive modes as bktaddjob.
**Eg:./bktdeletejob –buckId b1, ./bktdeletejob –buckId b1 –num_of_jobs 5**
bucketDB is similarly updated to reflected the current status of bucket after deletion.

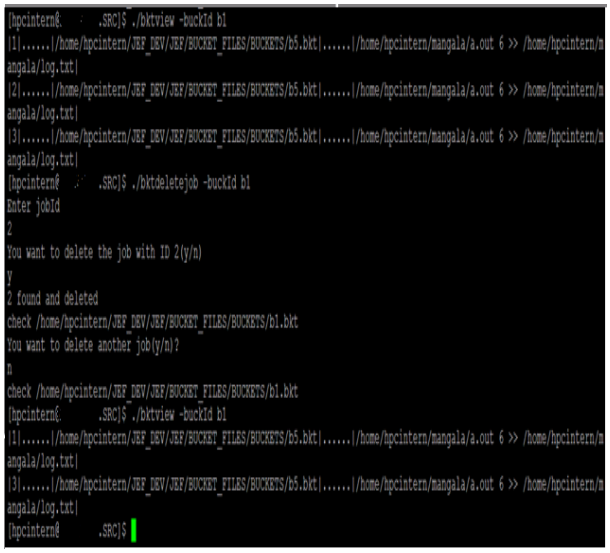


Figure 11 : Deleting jobs from bucket

7. **bktview** This prints the contents of the specified bucket and is used as:
**Eg:./bktview –buckId b1**

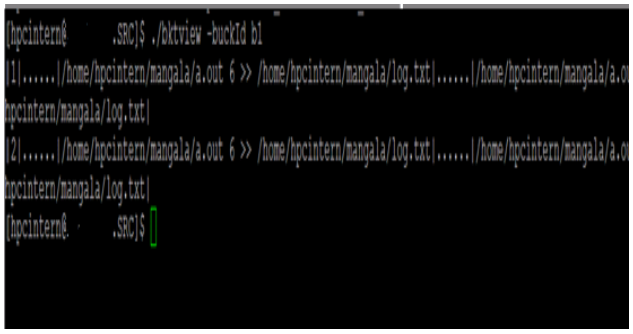


Figure: 12 Bucket view

8. **Scscript** This creates an executable form of bucket one at a time. These scripts are located at $JEF_PATH/ BUCKET_FILES/EXEC
**Eg: ./scscript –buckId b1**

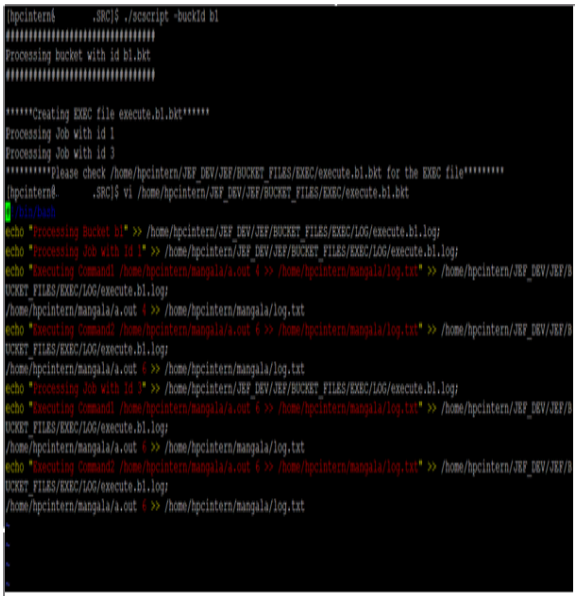


Figure: 12 Creating Exec file for bucket

9. **Sclaunch** It creates a master executable file with the executable scripts of a single/multiple buckets. This is also located at $JEF_PATH/BUCKET_FILES/EXEC. Now this file can be either executed directly or scheduled using CRON.
**Eg: ./sclaunch –buckId b1 –num_of_bkts 1** num_of_bkts option requires the number of buckets input to create the master executable file. Master executable file name is an optional input by user, by default it creates its own name. Once it creates the master executable file, it asks for an input by user if the file is to be executed directly or time schedule it using cron. A switch "f" overrides the above interaction with the user and executes it directly. If the user selects to schedule it using cron, it calls another script "scsched" which does scheduling.
**Eg: ./sclaunch –buckId b1 -num_of_bkts 1 –f** executes the script directly without requiring input from the user.
**Eg: ./sclaunch –buckId b1 - num_of_bkts 1 –f –time "31 16 5 7 4"** schedules the master script for the time specified in cron format without requiring an input by user during run time.


Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

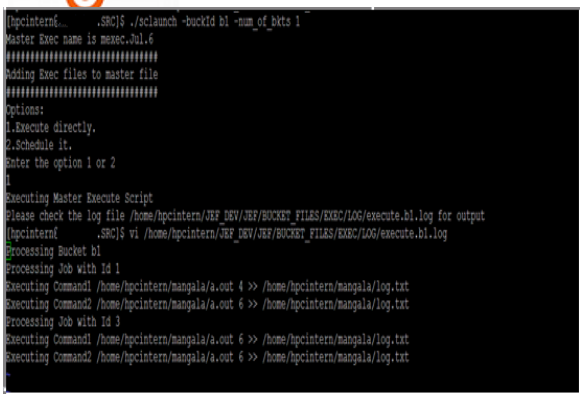



Figure: 13 Creating Master Exec file for bucket to execute

10. **scsched** It accepts master executable file and creates an executable file which is written into crontabalong with the time specified in cron format.
    **CRON[3]** is the time-based job scheduler in Unix-like computer operating systems. cron enables users to schedule jobs (commands ) run periodically at certain times or dates. Cron is driven by a *crontab* file, a configuration file that specifies shell commands to run periodically on a given schedule. The crontab files are stored where the lists of jobs and other instructions to the crondaemon are kept. Each line of a crontab file represents a job and is composed of a CRON expression, followed by a shell command to execute.

```
*  *  *  *  *  Command
|  |  |  |  |
|  |  |  |  +----- day of week (0 - 6) (0 is Sunday)
|  |  |  +-------- month (1 - 12)
|  |  +----------- day of month (1 - 31)
|  +-------------- hour (0 - 23)
+----------------- min (0 - 59)
```

Figure: 14 Cron Time Format

This is not open to users but called internally by sclaunch when required to schedule the master exec file using cron.

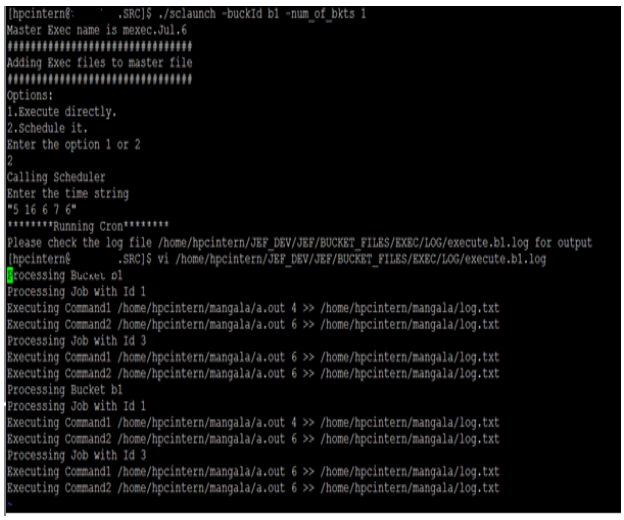



Figure: 15 Scheduling Master Exec file using Cron

## IV. CONCLUSION

This paper described the architecture of a generic framework for creation, management and execution of buckets directly or time schedule it using cron. JEF has been developed completely in perl and the desired outputs are validated. Future work may involve adding more functions to it such as: shuffling of jobs in the bucket, schedule it using a sophisticated scheduler which can handle priorities and sense upcoming buckets during run time etc.

## REFERENCES

1. http://en.wikipedia.org/wiki/Software_performance_testing#Methodology
2. http://en.wikipedia.org/wiki/Software_performance_testing
3. http://en.wikipedia.org/wiki/Cron
4. White Paper Performance Testing Methodology by Johann du Plessis
5. Teach Yourself Perl 5 in 21 days David Till


Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)