

Operation and Implementation of Random Variables in FileBench

Andrew W. Wilson
Sun Microsystems

Introduction

FileBench uses random distributions internally to select values used in a number of operations. Filesets are populated using three workload specified Gamma random distributions to select directory width, depth and file size for each directory and file created as part of the fileset. The I/O issuing flowops use uniform random distributions to select I/O buffer offsets into thread memory, while *read*, *write* and *aiowrite* flowops also can be directed to select offsets into a file from a uniform random distribution, and *appendfilerand* uses a uniform random distribution to select the amount to append to a file at each invocation. It would be a useful extension to FileBench if the workload model could specify the distribution types and perhaps additional parameters to use in each of these situations.

In addition to increasing the flexibility of the current uses of random number distributions, there are other values used in FileBench where a randomness option would be appropriate. One way to make a large number of attribute values potentially random is to use a feature of some simulation languages, known as *random* variables, which are variables that return a new value from a random distribution each time they are accessed. Hence this proposal to add a scheme for random variables to FileBench, which includes a mechanism for specifying the distribution to use in generating the values returned by the random variables. Since real world systems exhibit many forms of randomness in their behavior, having an efficient way to represent this in a modeling language is very important.

Thus, FileBench needs more flexibility in its current use of random distributions, and also needs a mechanism to extend the use of random distributions to other parameters. Using a common mechanism for all these situations would be ideal, however there are several approaches that could be taken.

This document proposes a random variable scheme for FileBench and optionally a random distribution definition scheme. A prototype that offers an extension of the current FileBench variable system using both random variables and random distribution entities has been created. This prototype will be described, along with several related alternatives. It is expected that other distributions will eventually be supported, and that the preliminary syntax specified here may change after review.

Declaring a Random Variable

There are at least two ways of directly creating random variables that would be consistent with current FileBench Workload Language syntax. If a separate Random Distribution Object is defined, then a third method of indirectly specifying a random variable by setting it to a named Random Distribution Object becomes possible. The three approaches are: Extending the *set* command to create a random variable and load its random distribution parameters, creating a new *define randvar* command to create a random variable and load its distribution parameters, and allowing the *set* command to assign a variable to a random distribution object. The first two approaches will be described next, then the third

will be described after Random Distributions Objects are introduced.

Through Extended “set” Command

A random variable could be declared using the same *set* command as is used to create or update the value of a normal variable. However, a random variable will include a *random* attribute, plus other attributes necessary to define the random distribution. Once declared to be random, a variable could not change back to being a normal variable. However, it can have its parameters changed by future *set* commands. It would also be possible to change a normal variable into a random one. A preliminary syntax for declaring a random variable is shown in Figure 1. Note in the example portion how a random variable *\$iosize* is declared, and then used with the *read* flowop to do random sized reads from the file *largefile*.

Doing a normal *set* operation on a variable declared as random should not change the variable, and a warning message should be generated. Random variables can be used anywhere normal variables are used to supply an integer. There is currently no concept of a random string variable.

Syntax:

```
set $<variable name> random[=<type>] [, seed=<value>] [, mean=<value>]
[, gamma=<value>] [, min=<value>] [, max=<value>]
```

Example:

```
set $iosize random, mean=4k, gamma=1000, min=512, max=16k
...
    flowop read name=rr, filename=largefile, iosize=$iosize
...
```

Figure 1: Syntax for and Example of Random Variables declared with the *set* command

Through a new “define randvar” command

Set is normally used to create or update a variable with a single value assignment. In FileBench, more extensive specifications are handled by the *define* command. Thus, it would seem reasonable to follow the same procedure here. The *set* command could still be used for changing a single parameter within the random variable's distribution though, by a modest extension of the *set* syntax.

As shown in Figure 2, a random variable would first be defined, which would create it and supply it with the parameters needed by its random distribution. It could then be used like any other variable, just as done with the extended *set* command approach. However, the *set* command could now be used to modify specific distribution parameters, by extending it with the classic name.parameter format. A big advantage of this is that it could be used with the existing FileBench perl wrapper to batch runs with different parameters in each run.

Syntax:

```
define randvar $<random variable name> [type=[uniform|gamma|table]]
[, seed=<value>] [, mean=<value>] [, gamma=<value>] [, min=<value>]
[, round=<value>] [, randsrc=[urandom|rand48]
[, pdftab={{<%>, <min value>, <max value>}, ...}

set $<random variable name . parameter>=<value>
```

Example:

```
define randvar name=$iosize mean=4k, min=1k, round=512
...
    flowop read name=rr, filename=largefile, iosize=$iosize
...
    Figure 2: Syntax for and Example of Random Variables declared with the define command
```

Random Distribution Object

As discussed in the introduction, there are a number of built-in uses of random numbers in FileBench, which need more specification flexibility. One approach would be to create a Random Distribution Entity and define it similarly to the way other entities are defined. Once defined, a Random Distribution Entity could be passed to the *define fileset* command and those flowops that currently use random numbers, as well as assigned to variables to make them random variables.

Defining a Random Distribution Object

Syntax:

```
define randdist name=<name> [, type=<distribution>] [, seed=<initial seed value>]
[, mean=<mean value>] [, mean=<gamma value>] [, min=<minimum value>]
[, round=<roundoff value>] [, source=[urandom|rand48]]
[, pdftab={{<%>, <min value>, <max value>}, ...}]
```

Example:

```
define randdist name=randdist1, mean=4k, gamma=1000, min=512,
max=16k
```

Figure 3: Syntax for defining a Random Distribution Entity

A *define* command is used, with the type of object specified as a *randdist*. Attributes are *name*, *type*, *seed*, *source*, *mean*, *gamma*, *min* and *max*. Figure 3 shows the proposed syntax and an illustrative example. These attributes would be adequate to cover the Gamma and Uniform distributions, but other attributes may be required for other types of distributions. One in particular that would be good to add

is a table driven distribution, where the probability density function (pdf) is supplied in tabular form. How to specify the pdf table is an open question at the moment. Figure 2 includes a proposed syntax for such a table when used with the *define randvar* command, and a similar syntax could be used here.

The *source* attribute refers to the source of random numbers used by the distribution. In the current FileBench implementation the random numbers are either generated by calls to `rand48()`, a standard pseudo random number generator, or obtained from hardware generators, through reads of the `/dev/random` pseudo device. Currently the gamma distribution always uses `rand48()` and the uniform distribution always uses `/dev/random`, but with this proposal, either source could be used by any distribution function.

The type attribute might initially specify *uniform* and *gamma*, but *normal*, *Weibull* and *table driven* are three other distributions which are frequently used in modeling work and could be included. The table driven approach allows specification of piece wise linear approximations of arbitrary distributions and thus can handle many situations where the distributions observed with real workloads cannot be modeled by any of the standard functions.

Setting Random Variables using Defined Random Distributions

If the concept of a Random Distribution Object is added to FileBench, then the distribution information can be specified there, and all the *set* command needs to do is reference the defined distribution. In order to distinguish the Random Distribution Entity's name from a simple string, the question mark (?) character is used at the beginning of the name. Figure 4 shows how this would be done.

Syntax:

```
set $<iosize>=?<random distribution name>
```

Example:

```
define randist name=randdist1, mean=4k, gamma=1000, min=1k,
round=512
...
set $iosize=?randdist1
```

Figure 4: Syntax for setting random variable to previously defined random distribution

Implementation

Three enhancements to FileBench allow the implementation of random variables. One enhancement is an extension to `var_integers` which adds a type field and consequently requires that they be accessed by an explicit subroutine call, the second similarly adds a new type to variables and a pointer to an extended data structure that holds information about the random distribution used by the variable. The third, which also supports random distribution objects is the `randdist_t` data structure which encapsulates the information needed by the desired random distribution function as used by random variables and other occurrences of randomness in FileBench.

The new, enhanced `var_integer` type is shown in figure 5. The first field is an enumeration type with the following values defined:

- `VIS_VALUE`
- `VIS_IND_VARVAL`
- `VIS_IND_RANDVAR`

Following the `v_type` field is a union of a `vinteger_t` (integer) an integer pointer and a random distribution pointer. If type is `VIS_VALUE`, then a 64 bit integer value will be stored in the union. If type is `VIS_IND_VARVAL`, then the integer pointer subfield points to the integer value in a normal variable. Finally, if the type is `VIS_IND_RANDVAR`, then the random distribution pointer subfield points to the random variable extension field that describes the random variable's distribution.

To use this new `var_integer_t` definition, values within the `var_integer` must be accessed through explicit functions. In many cases the old `var_integer` type was accessed through simple pointer dereferencing, and all such locations in the FileBench code where `var_integer` types are accessed have been updated to use the explicit function call. This has resulted in the addition of two new functions: a global `var_int_get()` function to access the `var_integer` and a local (to `vars.c`) `integer_alloc_var_ptr()` function to allocate a `var_integer` and set it to point to a supplied variable or random variable.

The random variables themselves are just ordinary variables with an

extension. However, to make this work an additional variable type has been added to distinguish between ordinary and random variables. The new type is `VAR_TYPE_RANDVAR`. Variables of this type point (using the `var_string` pointer) to a `randdist_t` object which contains routine pointers for fetching a new random value and a set of `var_integers` for various distribution function parameters. Currently only the Gamma distribution is supported. The random distribution entity is shown in Figure 6.

When an attribute of a FileBench entity is assigned to a random variable, the created `var_integer_t` is set to `VIS_IND_RANDVAR` type and when `var_int_get()` is called, the `rnd_get()` function from the random extension object is called with a pointer to the extension. Calling the `rnd_get()` function

old `var_integer_t`:

```
typedef vinteger_t *var_integer_t;
```

new `var_integer_t`:

```
typedef struct _var_int {
    vis_type_t v_type;
    union {
        vinteger_t intval;
        vinteger_t *intptr;
        struct _randdist *randptr;
    }v;
} *var_integer_t;
```

Figure 5: Old and new `var_integer_t`

```
int      (*rnd_get)(struct _randdist *, vinteger_t *);
void     (*rnd_set)(struct _randdist *, var_integer_t);
struct randdist *rnd_next;
char     *rnd_name;
var_integer_t rnd_seed;
var_integer_t rnd_mean;
var_integer_t rnd_gamma;
var_integer_t rnd_min;
var_integer_t rnd_round;
uint16_t  rnd_xi[3];
uint16_t  rnd_type;
```

Figure 6: Fields in the `randist_t` random distribution entity

generates a new random number using the parameters in the extension, and returned to the `var_int_get()` function which returns it to the caller.

Discussion of Approaches

To summarize, a Random Distribution Object has been proposed, which may be either explicitly or implicitly defined (that is, created as a side effect of defining a Random Variable). Then three ways of instantiating or defining a Random Variable have been proposed: using an extended *set* command, using an extended *define* command, or using a basic *set* command to assign it to a previously defined Random Distribution Object.

The motivation for an explicitly defined Random Distribution Object is that it might be more appropriate to use with existing instances of random distribution use in FileBench than would Random Variables. However, an analysis of the the six current uses suggests otherwise.

With filesets, random numbers are used to populate the fileset, determining the size of each file created, the width of each subdirectory, and the depth of a particular branch of the subdirectory tree. If fixed values are desired, then the gamma attributes must be set to zero. However, if these three parameters were brought out as explicit attributes, then fixed values could be directly specified using either integer values or normal variables, while random values could be specified by assigning the attributes to random variables. You might then specify a fileset as shown in Figure 7.

Random attributes example:

```
define randvar name=$filesize type=gamma, mean=16k, gamma=15000,
min=512, max=64k
```

```
define randvar name=$dirwidth type=gamma, mean=20, gamma=1500,
min=1
```

```
define randvar name=$dirdepth type=gamma, mean=4, gamma=1500,
min=0
```

```
define fileset name=bigfileset, filesize=$filesize,
dirwidth=$dirwidth, dirdepth=$dirdepth, entries=10000
```

Fixed attributes example:

```
define fileset name=bigfileset, filesize=16k, dirwidth=20,
entries=10000
```

Figure 7: Defining a fileset using random or normal variables.

With the seven flowops that explicitly generate I/O transfers: *read*, *write*, *aiowrite*, *readwholefile*, *writewholefile*, *appendfile*, and *appendfilerand*, uniformly distributed random numbers are used to select the memory buffer offset into thread memory. For the first three, *read*, *write* and *aiowrite*, uniformly distributed random numbers are also used to select locations within the file to write or read,

when random access within the file is requested. Finally, the `appendfilerand` flowop uses a uniform random distribution to select the amount to append with each invocation.

The thread memory buffer offset selection is hard coded into the flowops in the current implementation. Some new attribute, such as `memoffset` would need to be defined. It is not clear how useful a fixed offset would be, but there isn't any harm in allowing it. Thus there is an advantage assigning the `memoffset` attribute to a Random Distribution Entity, which enforces randomness, but a variable, generally a Random Variable, could also be used.

The selection of random file offset for those flowops that allow it is enabled with the `random` attribute, but otherwise cannot be controlled. One option would be to allow assignment of the `random` attribute to either a Random Distribution Object or a Random Variable. Of those two options, I think a Random Distribution Object makes more syntactic sense, but assignment to a Random Variable still seems reasonable.

Finally, with Random Variables, the `appendfilerand` flowop is no longer needed, as the `appendfile` flowop can just be used with its `iosize` attribute set to a Random Variable. However, `appendfilerand` should be kept around for now for backwards compatibility.

Current Proposal and Prototype

Based on the above arguments, I feel that adding Random Variables to the language with the `define randvar` command option is the most consistent and clean approach. This option would also include setting of particular random distribution parameters using the `set` command. The full syntax for both the `define randvar` and extended `set` commands is as shown in Figure 2. Note the addition of the `randtable` distribution type, which allows the specification of a probability function in tabular form. Each row of the table includes a percent (1-100), a minimum value and a maximum value. The percentages need to add up to 100. For a given row, it will be chosen on average *percent* of the time and a value uniformly selected between the maximum and minimum entries for the row will be generated.

The prototype implementation allocates a `randdist_t` object with each new random variable, but those objects are not visible within the language. Using a common object to hold random distribution information throughout FileBench provides a consistent internal representation, thus simplifying the implementation of randomness within FileBench, and facilitating extensions to other useful distribution types. The old syntax for filesets and the I/O generating flowops is still being kept as the default approach for backwards compatibility, but, as time permits, the specification and implementation of random distributions in those will be modified to use `randdist_t` objects and be settable through random variables. A use of a random variable to select the filesize for each file within a fileset is implemented in the prototype.

An example workload which utilizes a pair of table driven random variables, one for the I/O size and one for the file size of each file in the fileset is shown in Figure 8. This workload rotates through the `$nfiles` files of the fileset `fileset1`, which has been formed with a distribution of file sizes specified by the random variable `$filesize`. On each iteration, the defined thread opens a file from the fileset, reads the entire amount using an I/O size specified by the random variable `$iosize`, then closes the file and continues on to the next file. Note that if the randomly chosen I/O size is larger than the selected file's size, a single read of the whole file will be performed. The output from a run of the workload is shown

in Figure 9.

```

#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2008 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# ident "@(#)netreader.f          1.0      01/24/08 SMI"

set $dir=/tmp
set $nfiles=10000
set $nthreads=1

define randvar name=$filesize, randsrc=rand48, min=1k, round=1k, randtable={
  {40, 1k, 1k},
  {30, 2k, 8k},
  {20, 8k, 32k},
  {10, 32k, 128k}}

define randvar name=$iosize, randsrc=rand48, min=512, round=1k, randtable={
  {50, 512, 1k},
  {25, 1k, 8k},
  {25, 8k, 64k}}

define fileset name=fileset1,path=$dir,size=$filesize,prealloc,paralloc,entries=$nfiles,dirwidth=20

define process name=netread,instances=1
{
  thread name=netrd-thread,memsize=10m,instances=$nthreads
  {
    flowop openfile name=netrd-open1,filename=fileset1,directio=0,fd=1
    flowop readwholefile name=netrd-rwf1,iosize=$iosize,fd=1
    flowop closefile name=netrd-closet1,fd=1
    flowop eventlimit name=netrd-rate
  }
}

echo "netreader Version 1.0 IO personality successfully loaded"
usage "Usage: set \ $dir=<dir>"
usage "      set \ $nfiles=<value> defaults to $nfiles"
usage "      set \ $filesize=<size> defaults to $filesize"
usage "      set \ $iosize=<value> defaults to $iosize"
usage "      set \ $nthreads=<value> defaults to $nthreads"
usage "      run runtime (e.g. run 60)"

```

Figure 8: Example netreader.f workload

Conclusion

The addition of Random Variables and their use to specify all instances of randomness in FileBench is useful and desirable and should be done as soon as possible.

```

filebench> load netreader
111848: 3.649: Usage: set $dir=<dir>
111848: 3.649:      set $nfiles=<value>      defaults to 10000
111848: 3.650:      set $filesize=<size>      defaults to tabular random var
111848: 3.650:      set $iosize=<value>      defaults to tabular random var
111848: 3.650:      set $nthreads=<value>      defaults to 1
111848: 3.650:      run runtime (e.g. run 60)
filebench> set $dir=/export/home/tmp
filebench> run 60
111848: 16.362: Creating/pre-allocating files and filesets
111848: 16.883: Fileset fileset1: 10000 files, avg dir = 20, avg depth = 3.1,
mbytes=13242
111848: 23.057: Removed any existing fileset fileset1 in 7 seconds
111848: 23.658: Creating fileset fileset1...
111848: 1267.692: Preallocated 10000 of 10000 of fileset fileset1 in 1245
seconds
111848: 1267.692: waiting for fileset pre-allocation to finish
111848: 1268.640: Starting 1 netreader instances
111853: 1269.700: Starting 1 filereaderthread threads
111848: 1272.710: Running...
111848: 1333.210: Run took 60 seconds...
111848: 1333.211: Per-Operation Breakdown
netrd-rate          0ops/s   0.0mb/s   0.0ms/op   0us/op-cpu
netrd-close1       24ops/s   0.0mb/s   0.0ms/op   32us/op-cpu
netrd-rwfl         24ops/s  32.7mb/s  40.6ms/op  10565us/op-cpu
netrd-open1        24ops/s   0.0mb/s   0.4ms/op   343us/op-cpu

111848: 1333.212:
IO Summary:          4379 ops  72.4 ops/s, (24/0 r/w)  32.7mb/s,  59857us cpu/op,
41.1ms latency
111848: 1333.212: Shutting down processes
filebench> quit

```

Figure 9: Output from run of netreader.f example