ResQue
User's Manual

Version 0.1

October 15, 2012

Sudharshan Varadarajan, H. M. N. Dilum Bandara, and Anura P. Jayasumana
Computer Networks Research Laboratory
Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO 80523.
www.cnrl.colostate.edu

Contents

1 Introduction

1.1 Overview

Multi-attribute **Res**ource and **Que**ry (ResQue) generator can be used to generate large synthetic traces of computing resources and range queries. Such traces are useful in large-scale performance studies of resource discovery systems, job schedulers, etc., in collaborative peer-to-peer systems, volunteer computing, and grid and cloud computing. ResQue preserves the statistical properties of real-world computing resources such as distributions of attributes, complex correlation between static and/or dynamic attributes, contemporaneous correlation between dynamic attributes, and popularity of attributes. Users may use the provided datasets or their own ones as the basis to generate large synthetic traces. Several tools are also provided to simplify the conversion of other datasets to the format supported by ResQue.

1.2 Related Publications

1. H. M. N. D. Bandara and A. P. Jayasumana, "On characteristics and modeling of P2P resources with correlated static and dynamic attributes," In Proc. IEEE Globecom '11, Dec. 2011.
2. H. M. N. D. Bandara and A. P. Jayasumana, "On characteristics and generation of multi-attribute resources and queries with correlated attributes," In review.
3. H. M. N. D. Bandara and A. P. Jayasumana, "Characteristics of multi-attribute resource/queries and implications on P2P resource discovery," In Proc. 9th ACS/IEEE Int'l Conf. on Computer Systems and Applications (AICCSA '11), Dec. 2011.
4. H. M. N. D. Bandara and A. P. Jayasumana, "Evaluation of P2P resource discovery architectures using real-life multi-attribute resource and query characteristics," In Proc. IEEE Consumer Communications and Networking Conf. (CCNC '12), Jan. 2012.
5. H. M. N. D. Bandara and A. P. Jayasumana, "Multi-attribute resource and query characteristics of real-world systems and implications on peer-to-peer-based resource discovery," In review.
6. H. M. N. D. Bandara, "Enhancing collaborative peer-to-peer systems using resource aggregation and caching: A multi-attribute resource and query aware approach," PhD Dissertation, Colorado State University, Fall 2012.
7. J. C. Strelen, "Tools for dependent simulation input with copulas," In Proc. 2nd Int. Conf. on Simulation Tools and Techniques, Mar. 2009.

1.3 License

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

1.4 Points of Contact

Reach us at www.cnrl.colostate.edu/pages/contactus.html

Please inform us about any errors, bugs, and feature requests.

1.5 Outline

Section 2 provides a brief overview about the resource and query generation. Step by step instructions on resource and query generation are presented in Section 3. Section 4 presents the list of attributes supported by current datasets. Section 5 discusses the syntax of the input and output file formats and how other datasets can be used with ResQue. Instructions on completing ResQue and common error messages are given in Section 6.

2. System Summary

2.1 Multi-Attribute Resource Generation

Each resource *r* is defined as follows:

$$r = \left(a_1 = v_1, a_2 = v_2, \ldots, a_i = v_i\right)$$

Each attribute a_i has a corresponding value $v_i \in \mathbf{D_i}$ that belongs to a given domain $\mathbf{D_i}$. $\mathbf{D_i}$'s are typically bounded and may be continuous or discrete. For example, free CPU (*CPUFree*) is continuous and number of CPU cores (*NumCores*) is discrete. This version of ResQue does not support categorical attributes such as CPU architectures and operating systems. A multi-attribute Resource Specification (RS) with such a set of attributes may look like the following:

$$RS = \left(CPUSpeed = 2.4GHz, NumCores = 2, CPUFree = 53\%, MemFree = 1071MB\right)$$

Resource generation module combines the empirical-copula-based static attribute generation and time-series-library-based dynamic attribute generation (see Fig. 1)**.** Based on the input dataset, the tool generates synthetic traces of nodes with static and/or dynamic attributes. Users may generate resources for any subset of the attributes that are present in the input dataset.

Random vectors of static attributes are generated using empirical copulas [7]. Copulas are functions that couple the multivariate distribution functions to their marginal distributions. Empirical copulas are useful while analyzing data with complex and/or unknown underlying distributions. We use empirical copulas to generate vectors of static attributes, as the joint distribution is unknown and complex. Empirical copulas enable us to use the empirical data directly while generalizing ResQue to any multivariate dataset regardless of its dependence structure.

First, all the active nodes at a given time instance is sampled for their static attributes. The instantaneous values of dynamic attributes are also sampled as some performance studies need the instantaneous values of dynamic attribute not the time series. Second, the marginal distribution of each attribute is then transformed to a uniform random variable $\sim U(0, 1)$, e.g., using Kernel smoothing density estimation. Third, empirical copula is calculated using the method described in [7]. Fourth, dependent random numbers are then generated from the multivariate copula. Finally, random numbers are transformed back to desired marginal distributions using inverse transformation techniques, e.g., using estimated empirical distribution functions. If the attribute value is continuous, linear interpolation may be used to generate in-between values while performing the inverse transformation. Empirical distribution functions can be used for discrete valued attributes (e.g., *NumCores*). We use *pwlCopula* [7] MATLAB tool to generate the static attributes. See [1-2, 6-7] for additional details.

Time varying dynamic attribute values cannot be drawn randomly from marginal distributions as the time series of some of the dynamic attributes have a specific structure [1-2, 6]. Moreover, the contemporaneous correlation between two time series needs to be preserved. Therefore, we build a library of time-series segments by identifying specific temporal patterns

exhibited by dynamic attributes. This is sufficient, as our goal is to preserve the temporal variation of an attribute and its contemporaneous correlation with other attributes. We select free memory (*MemFree)* as the attribute based on which to partition the time series due to its distinguishable pattern and high autocorrelation. We proposed two mechanisms to identify the structural changes in time series and detailed are provided in [1, 2, 6].

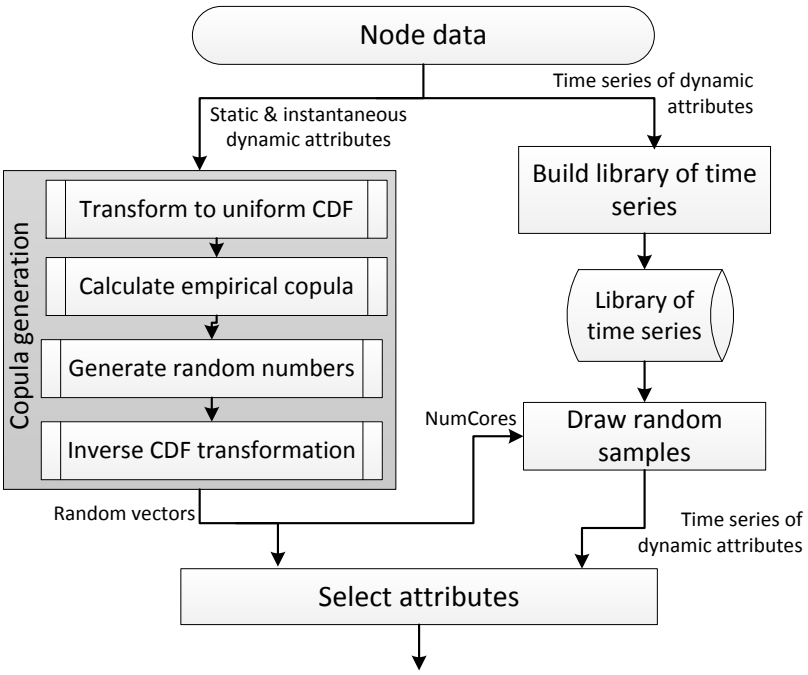

Figure 1 – Flowchart of resource generation.

Once the structural changes in the *MemFree* time series are identified, time series corresponding to rest of the dynamic attributes are split at the same breakpoints. Resulting multivariate time series segments are then collected to form a library. For generating dynamic attribute values, multivariate time series segments are drawn randomly from the library. Longer sequences are generated by concatenating one randomly drawn segment to another. Breaking all the time series of a node at the same point and replaying them together preserve the contemporaneous correlation among attributes. Randomly mixing time series segments corresponding to busy and idle periods is acceptable in moderately busy systems such as PlanetLab and enterprise computing where distribution of attributes tend to be stable over several hours to a few weeks [6]. However, such random mixing is not suitable in grid and cloud computing where the entire system or a large fraction of it oscillates between busy and idle periods [2, 6]. In practice, one may want to build a synthetic trace where the system is busy during given time ranges, moderately busy in another set of time ranges, and idle in the remaining times. For example, one may want to build a trace where the system is idle within the first 6 hours, it then remain busy during the next 12 hours (e.g., due to arrival of bag of tasks), and then becomes moderately busy for another 6 hours. Such traces are useful in determining the adaptability of resource discovery solutions. Such constraints are accomplished by grouping the time-series segments in the library according to a given attribute. For example, based on the average *CPUFree*, *MemFree*, and/or *TxRate* values time

series segments can be labeled as *idle* or *busy* (ResQue currently supports only *CPUFree*. Users may modify the source code if they want to use other attributes). Depending on the user requirements, time segments can be randomly drawn from only the subset of segments that are labeled as *idle* or *busy*. Moderate loads can be generated by randomly drawing time series segments marked as *idle* or *busy* (if the number of *idle* and *busy* samples is similar, otherwise weights may be adjusted to get a similar number of samples from each group). ResQue allows such weighted selection of time series by allowing users to specify thresholds for idle, moderately busy, and busy loads.

As the static and dynamic attributes are correlated, it is essential to establish the dependency between them. For example, a node with large *NumCores* typically has higher *CPUFree* values [1-3, 5-6]. Therefore, time-series segments in the library are grouped according to the *NumCores* of the corresponding node. Consequently, given the *NumCores* generated from empirical copula, the dependency between static and dynamic attributes can be established by randomly drawing time-series segments from the corresponding group. This is sufficient to establish the correlation as correlation between other static attributes and dynamic attributes is not strong [1-2, 6]. ResQue allows the use of other attributes to establish the correlation, but make sure there are enough time series segments for the static attribute that you selected. If not, ResQue will print a warning message.

2.2 Multi-Attribute Range Query Generation

A multi-attribute range query q is defined as follows:

$$q = \left(m_q, a_1 \in [l_1, u_1], a_2 \in [l_2, u_2], \ldots, a_i \in [l_i, u_i]\right)$$

where, $m_q \in \mathbf{Z}^+$ specifies the required number of resources and $a_i \in [l_i, u_i]$ specifies the desired range of attribute values (l_i and u_i are lower and upper bounds, respectively). A query that requests for five computing nodes with a given CPU speed, free memory, and operating system may look like the following:

$$q = \left(m_q = 5, CPUSpeed \in [2.0GHz, 3.0\,\text{GHz}], NumCores = 1,\ MemFree \in [256MB, 512MB]\right)$$

The set of attributes specified in a query may contain only a subset of the attributes that are used to describe resources. When $l_i = u_i$, for all the attributes in q it is referred to as a *point query*. In practice, attributes in a query may specify a mixture of point and range values. Unspecified attributes are considered as "don't care". q is referred to as a *single-attribute query* when it specifies only one attribute and it is referred to as a *multi-attribute query* when it specifies more than one attribute.

Multi-attribute range queries are generated using a Probabilistic Finite State Machines (PFSM). Suppose following three multi-attribute queries are given as the basis to generate synthetic queries (attribute values are ignored to simplify the discussion):

$Q_1 = \{CPUSpeed\}$

Q_2 = {*MemFree*, *1MinLoad*}
Q_3 = {*MemFree*, *CPUSpeed*, *TxRate*}

Suppose Q_2 appeared twice and Q_1 and Q_3 each appeared once. We build a PFSM to capture the popularity of attributes, number of attributes in a query, and occurrences of attribute pairs, triplets, etc. Attributes are interpreted as a set of states and attribute co-occurrences are interpreted as state transitions weighted by their frequency of occurrences. However, multi-attribute queries do not have well defined *START* and *FINISH* states. Therefore, we assumed virtual *START* and *FINISH* states, and then interpreted the first attribute specified in a query as a transition from the *START* state and the last attribute in the query as a transition into the *FINISH* state. Fig. 2 depicts the corresponding PFSM for the above three queries. Following distinct queries can be generated using this PFSM:

q_1 = {*CPUSpeed*}	1/8
q_2 = {*CPUSpeed*, *TxRate*}	1/8
q_3 = {*MemFree*, *1MinLoad*}	1/2
q_4 = {*MemFree*, *CPUSpeed*}	1/8
q_5 = {*MemFree*, *CPUSpeed*, *TxRate*}	1/8

Their probability of occurrence is indicated on the right. PFSM generates two queries q_2 and q_4 that was not among the original queries. There queries are also valid as there is a possibility of specifying *CPUSpeed* with *TxRate* and *MemFree*. Therefore, by applying a PFSM we can also generate many queries that are likely to occur in practice. Ranges of attribute values defined in queries can be represented as a set of sub-states. For example, two queries with *CPUSpeed* ∈ [1.5, 3.0] and *CPUSpeed* ∈ [2.0, MAX] can be defined as two sub-states within the main state *CPUSpeed*.

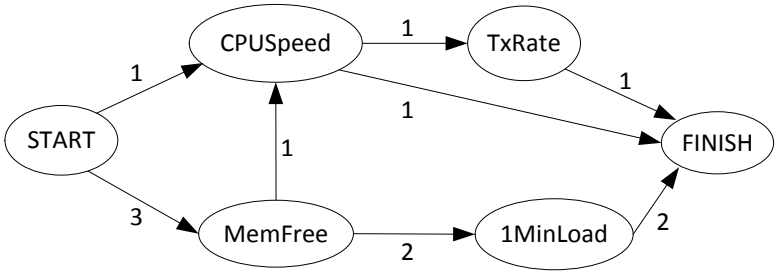

Figure 2 – Probabilistic finite state machine for queries Q_1, Q_2, and Q_3.

Suppose the attributes in the original query Q_2 is swapped as Q_2 = {*1MinLoad*, *MemFree*}. This is possible as it is not necessary to specific attributes in a particular order. Then the corresponding PFSM is given in Fig. 3. This PFSM is slightly different from Fig. 2 and produces the following set of distinct queries:

q_1 = {*CPUSpeed*}	1/8
q_2 = {*CPUSpeed*, *TxRate*}	1/8
q_3 = {*MemFree*, *1MinLoad*}	1/3
q_4 = {*MemFree*, *CPUSpeed*}	1/24

q_5 = {*MemFree*, *CPUSpeed*, *TxRate*} 1/24
q_6 = {*MemFree*} 1/6
q_7 = {*1MinLoad*, *MemFree*, *CPUSpeed*} 1/12
q_8 = {*1MinLoad*, *MemFree*, *CPUSpeed*, *TxRate*} 1/12

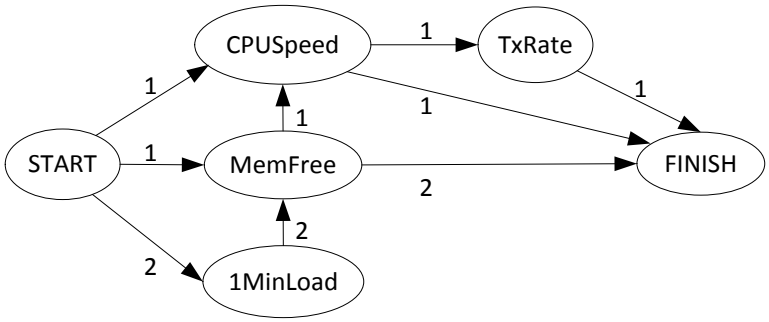

Figure 3 – Probabilistic finite state machine for queries when attributes in Q_2 is swapped.

It produces three new queries q_6 to q_8 in addition to the ones generated by the PFSM in Fig. 2. This is a consequence of not having well defined *START* and *FINISH* states in queries. Therefore, resulting PFSM is sensitive to how the states are coded. While this could result in generation of queries with invalid combinations of attributes, it also offers the opportunity to generate different mixtures of queries by using different coding conventions. For example, in addition to coding attributes based on the order they appear in queries, attributes in a query may be shuffled randomly or sorted in the ascending or descending order before building the PFSM. The problem of generating invalid queries can be handled by either ignoring those queries ones they are generated or modifying the PFSM to prevent the generation of such queries. In practice, modifying a large PFSM without reducing the possibility of generating other potential queries is not straight forward. Therefore, we recommend discarding invalid queries after they are generated.

3. Getting Started With ResQue

The latest version of ResQue can be downloaded from http://www.cnrl.colostate.edu/Projects/CP2P/. You need to also download at least one of the datasets or create your own one. Some of the utilities provided will be useful in converting your data to the format supported by ResQue. You need to have access to GUI version of MATLAB to run ResQue (ResQue is developed using MATLAB version 2012a). If you have access to the MATLAB compiler, you may compile ResQue using the instructions given in Section 6.1. Otherwise it can be run from the source. Following instructions are based on the *Dummy* dataset.

Step 1 Double click on the ***ResQue.fig*** file to start ResQue. This will open up the ResQue form (see Fig. 4).

Step 2 Pick one of the following options depending on what you want to generate:

- *Generate Resources* – Generates synthetic resources by combining the static and/or dynamic attributes of the nodes.
- *Generate Queries* – Generates synthetic queries based on the probabilistic finite state machine.

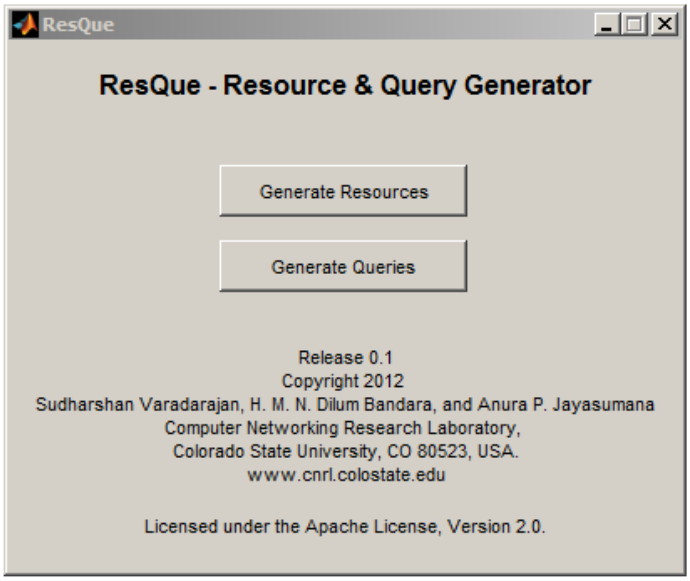

Figure 4 – ResQue startup form.

3.1 Multi-Attribute Resource Generation

Step 1 Click on the ***Generate Resources*** button on the ResQue form. This will open up the resource generation form (ResGen) in Fig. 5.

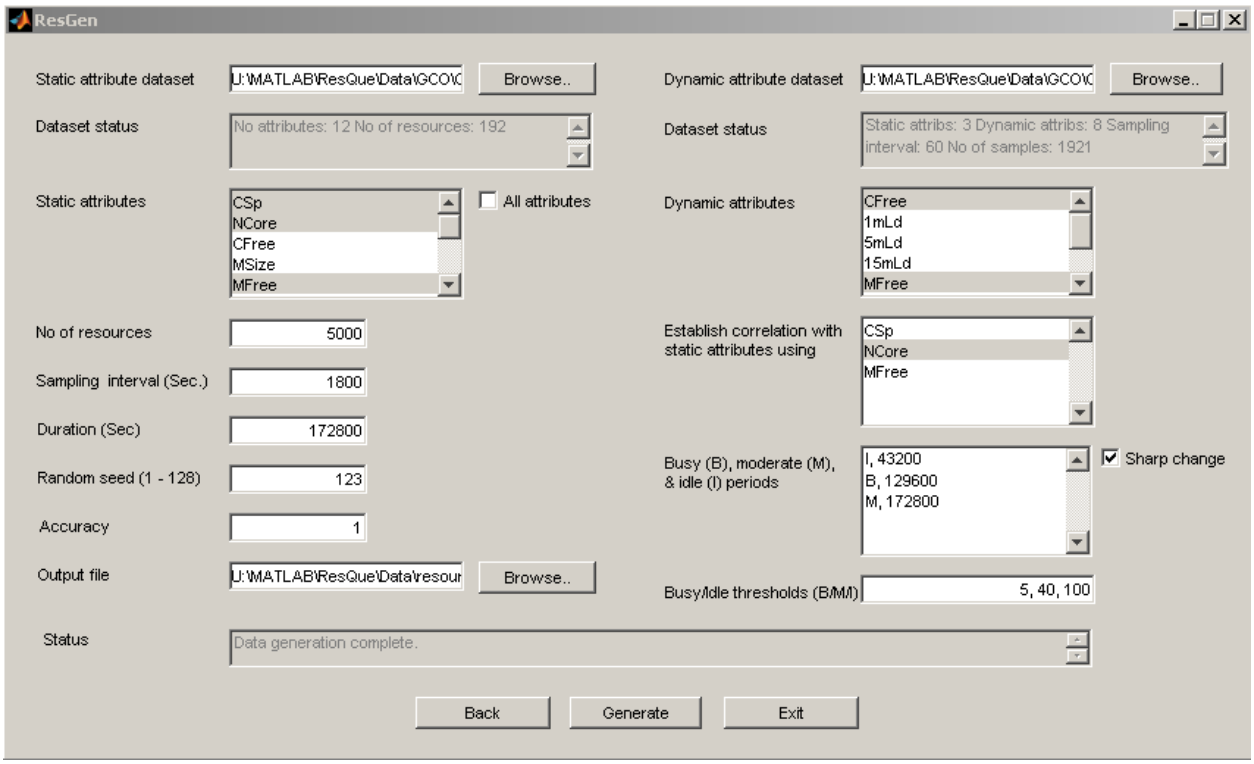

Figure 5 – Resource generator.

Step 2 Select the static attribute dataset as the basis to generate static attributes. Table 1 lists the description of each of the available options. If you do not want to generate static attribute you may skip to **Step 5**.

For this example, we will use the **Dymmy_Static.txt** file (included in the Data folder) by clicking on the **Browse…** button. Depending on the number of entries in the input file and speed of your machine it may take few seconds to a few minutes.

Once the file is properly read, **Dataset status** label will show the number of attributes and resources in the file. You will be informed of any errors. List of common error messages is given in Section 6.2.

Step 3 Select the static attributes that you want to generate from the **Static attributes** list box. To select multiple entries press **Control** button while selecting.

Each attribute that you select will be added to the **Establish correlation with static attributes using** the list box. One of these attributes can be used to establish the correlation with dynamic attributes. You may ignore them if you do not want to generate dynamic attributes or do not want to explicitly define the correlation between static and dynamic attributes.

Table 1 – List of options available while generating resources.

Label	Description	Valid Inputs
Static attribute dataset	File with static attributes of nodes to use as the basis to generate data	Properly formatted file (see Section 5.1.1)
Dynamic attribute dataset	Time series library file with dynamic attributes to use as the basis to generate data	Properly formatted file (see Section 5.1.2)
Static attributes	Static attributes to generate	Pick from list
All attributes	If selected, all attributes in the original data will be used while generating data using empirical copula	
Dynamic attributes	Dynamic attributes to generate	Pick from list
No. of resources	How many resources/nodes to generate	≥ 1 (integer)
Sampling interval (Sec.)	Sampling interval for dynamic attribute. Apply only to dynamic attributes. Values in seconds	Integer multiply of sampling interval in time-series library
Duration (Sec.)	Up to what time data to be generated. Apply only to dynamic attributes. Values in seconds	≥ 1 (integer)
Random seed	Random seed to use while generating data	1-128 (integer)
Accuracy	Used by pwlCopula to determine the granularity. Accuracy increase with larger numbers	≥ 1 (integer). Must be ≤ number of resources in static attribute dataset
Establish correlation with static attributes using	Attribute to use to establish the correlation with static attributes. Make sure selected attribute is actually available in time series library	
Busy (B), Moderate (M), & Idle (I) periods	Optional. Specify when to generate busy, moderately busy, & idle periods. Use the format shown in Fig. 5	<B/I/M, time (integer)>
Sharp change	Optional. Suddenly change from one state to another. Otherwise, the change will be gradual	
Busy/Idle thresholds (B/M/I)	Thresholds used to specify busy (B), moderately busy (M), & idle (I) samples	3 values between 1-100 separated by a comma

Step 4 Fill the **No. of resources**, **Random seed**, and **Accuracy** textboxes with the appropriate values. If you are also generating dynamic attributes also fill the **Sampling interval (Sec.)** and **Duration (Sec.)** textboxes.

Step 5 Next, select the dynamic attribute dataset as the basis to generate time series of dynamic attributes. If you do not want to generate dynamic attributes you may skip this.

For this example, we will use the **Dymmy_Dyanmic.txt** file by clicking on the **Browse…** button. Depending on the no of entries in the input file and speed of your machine it may take few seconds to tens of minutes. Once the file is successfully read, **Dataset status** will show the number of static and dynamic attributes in the file as well as sampling interval and number of samples (i.e., time series segments).

Step 6 Select the dynamic attributes that you want to generate from the **Dynamic attributes** list box. To select multiple entries press **Control** button while selecting.

Step 7 If you are generating both the static and dynamic attributes and would like to establish correlation between them select one of the attributes in the **Establish correlation with static attributes using** the list box.

Step 8 Optional. The default state is *moderately busy.* Therefore, while generating dynamic attributes time series segments are randomly selected from the time series library. You may generate idle and busy periods (for the entire system) by selecting a subset of the time series segments based on their average *CPUFree* values. Otherwise skip to **Step 10**. Such periods can be specified using the **Busy (B), Moderate (M), & Idle (I) periods** textbox. Add each entry in a separate line using the following format:

B/M/I, time

Following examples shows how to define the periods. Suppose we want the first 6 hours to be idle, next 12 to be busy, and another 6 hours to be moderately busy. Then the periods can be defined as follows:

I, 21600
B, 64800
M, 86400

If you do not explicitly specify the end time, remaining time is considered to be moderately loaded. If you want time series to rapidly change from one state to another select the **Sharp change** checkbox.

Step 9 Specify thresholds to classify time series segments as busy (B), moderately busy (M), & idle (I). Enter 3 values in the range 1-100 separated by a comma. Each value should be larger than the previous value. For example, one may specify

5, 40, 100

Step 10 Enter the name of the file to save the generated data in the **Output file** textbox. You may also use the **Browse…** button to navigate to a different folder.

Step 11 Double-check all the entries. Then click the **Generate** button to generate data.

Then ResQue will be asked how to convert the static attributes back to the original marginal distributions (see Fig. 6). If the attribute value is continuous, linear interpolation may be used to generate in-between values while performing the inverse transformation (e.g., *CPUSpeed* and *DiskSize*). Empirical distribution functions may be used for discrete valued attributes such as *NumCores*.

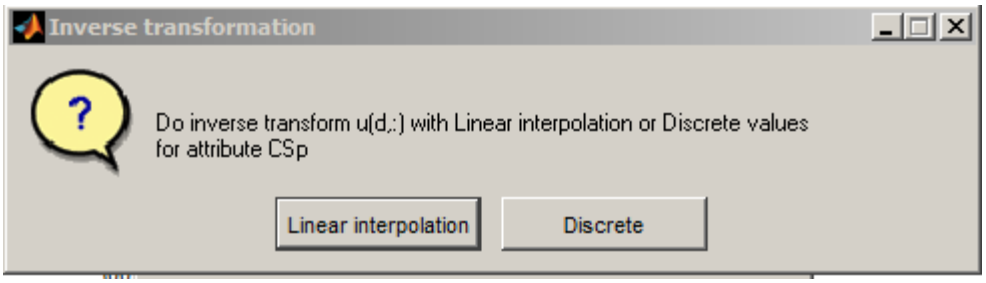

Figure 6 – Inverse transformation of generated attributes.

Depending on the number of resources to generate, sampling interval, duration, and speed of your machine it may take from a few seconds to several hours to generate the data. **Status** label indicates the current status of data generation. You may check the progress of the generated file using a command like *tail* in Linux or by opening it in the read only mode.

Note Once the data are generated you are encouraged to validate those using simple statistical tests such as mean, standard deviation, and by plotting cumulative distributions. Depending on the attributes you may also use a Kolmogorov–Smirnov (KS) test to validate the data. Some of the methods we used to validate the data are given in [1, 5-6].

3.2 Query Generation

Step 1 Click on the ***Generate Queries*** button on the ResQue form. This will open up the resource generation form (QueryGen) in Fig. 7.

Step 2 Select the query dataset (with the list of state transitions and their frequencies) as the basis to generate queries.

For this example, we will use the **Dymmy_Queries.txt** file (included in the Data folder) by clicking on the **Browse…** button. Depending on the no of entries in the input file and the speed of your machine it may take few seconds to a few minutes.

Once the file is read, **Status** label will indicate the number of attributes in the file. If the file already has the CDF of the number of resources requested by a query, it will be indicated as well. List of common error messages is given in Section 6.2.

Step 3 Specify the **No of queries** to generate. Table 2 lists the description of each of the available options.

Step 4 Then select one of the **No of resources per query** radio buttons to specify the number of resources requested by a query. See Table 2 for available options.

Step 5 Select the attributes to use while generating queries using the **Attributes to use** list box**.** To select multiple entries press **Control** button while selecting.

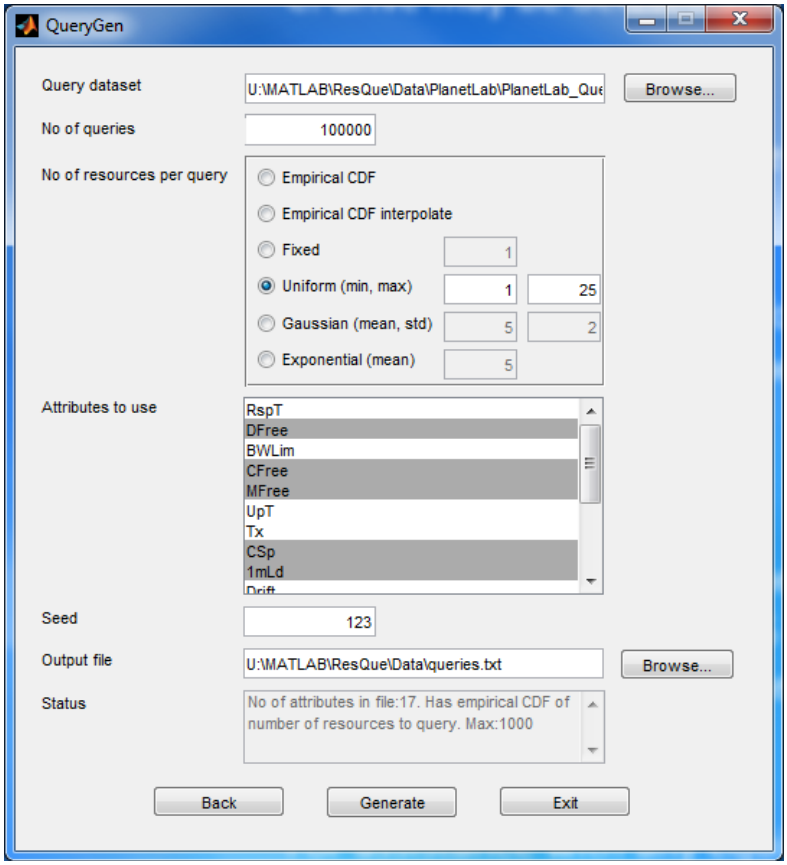

Figure 7 – Query generator.

Step 6 Set the **Random seed** and name of the file to save the generated data in the **Output file** textbox. You may also use the **Browse…** button to navigate to a different folder.

Step 7 Double-check all the entries. Then click the **Generate** button to generate the queries. Depending on the number of queries to generate, selected attributes, and speed of your machine it may take from few seconds to hours to generate the queries. The **Status** label indicates the current stats of query generation.

Note Once the queries are generated you are encouraged to validate those using the approaches described in [5-6].

Table 2 – List of options available while generating queries.

Label	Description	Valid Inputs
Query dataset	Dataset to be used as the basis to generate range queries	One of the given query files or user generated ones
No of queries	Number of queries to generate	≥ 1

No of resources per query	No of resources requested by each query. This is the m_q defined in Section 2.2. One the following options may be selected
	• Empirical CDF – Use the data from empirical CDF, if given in the input data file
	• Empirical CDF interpolate – Use the data from empirical CDF if given in the input data file while interpolating to generate in between values
	• Fixed – Use a constant values
	• Uniform (min, max) – Generate m_q based on the given uniform distribution. Define the range of values using the minimum and maximum. Real values will be rounded to integers
	• Gaussian(mean, std) – Generate m_q based on the given Gaussian distribution. Define mean and standard deviation. Real values will be rounded to integers
	• Exponential(mean) – Generate m_q based on the given exponential distribution. Define mean. Real values will be rounded to integers
		• Fixed – ≥ 1
		• Uniform
		  ○ min ≥ 1
		  ○ max ≥ min
		• Gaussian
		  ○ mean ≥ 0
		  ○ std > 0
		• Exponential
		  ○ mean ≥ 0

Attributes to use	Subset of the attributes to use.	Pick at least 1
Seed	Random seed	1-128 (integer)
Output file	Where to save the generated queries	

4 Attributes in Available Datasets

Tables 3 to 6 lists the names of attributes used to characterize resources, their data types, and units. Attribute names are abbreviated to reduce the file size of both input and output data. You may use other attributes as far they represent an integer or floating point value that is not a category or a type.

Table 3 – Attributes of PlanetLab nodes.

Attribute	Description	Data Type	Units
1mLd	1 minute average of CPU load	Float	-
5mLd	5 minute average of CPU load	Float	-
15mLd	15 minute average of CPU load	Float	-
CFree	Free CPU	Float	%
CSp	CPU speed	Float	GHz
DFree	Free disk space	Float	GB
DSize	Disk size/capacity	Float	GB
MFree	Free memory	Float	%
MSize	Memory size	Float	GB
NCore	No of CPU cores	Integer	-
Rx	Data receiver rate	Float	Bps
Tx	Data transmission rate	Float	Bps

Table 4 – Attributes of SETI@home nodes.

Attribute	Description	Data Type	Units
Cache	Cache size of CPU	Float	MB
CSp	CPU speed	Float	GHz
DFree	Free disk space	Float	GB
DSize	Disk size/capacity	Float	GB
MSize	Memory size	Float	GB
NCore	No of CPU cores	Integer	-
PIops	CPU performance based on Dhrystone (integer) benchmark	Float	-
PFpops	CPU performance based on Whetstone (floating-point) benchmark	Float	-
Rx	Data receiver rate	Float	bps
Swap	Size of swap memory	Float	GB
Tx	Data transmission rate	Float	bps

Table 5 – Attributes of GCO nodes.

Attribute	Description	Data Type	Units
1mLd	1 minute average of CPU load	Float	-
5mLd	5 minute average of CPU load	Float	-
15mLd	15 minute average of CPU load	Float	-
CSp	CPU speed	Float	GHz
CFree	Free CPU	Float	%
DFree	Free disk space	Float	GB
DSize	Disk size/capacity	Float	GB
MFree	Free memory	Float	GB
MSize	Memory size	Float	GB
NCore	No of CPU cores	Integer	-
Rx	Data receiver rate	Float	bps
Tx	Data transmission rate	Float	bps

Table 6 – Attributes of CSU nodes.

Attribute	Description	Data Type	Units
1mLd	1 minute average of CPU load (Linux only)	Float	-
5mLd	5 minute average of CPU load (Linux only)	Float	-
15mLd	15 minute average of CPU load (Linux only)	Float	-
CSp	CPU speed	Float	GHz
CFree	Free CPU	Float	%
DFree	Free disk space	Float	GB
DSize	Disk size/capacity	Float	GB
MFree	Memory size	Float	GB
MSize	Free memory	Float	GB
NCore	No of CPU cores	Integer	-

5 File Formats

5.1 Input File Formats

Any line starting with hash symbols "#" is a comment.

5.1.1 Static Attributes

First list the attribute names. Start with **ATTNAMES** command then list the attribute names in the same sequence as data separated by a tab. For example:

ATTNAMES<tab>CSp<tab>NCore<tab>MSize

Then use the **DATA** command to indicate that we are providing data. Next, enter attributes of nodes, each in a separate line, following the same order given in **ATTNAMES**. For example:

DATA
2.1<tab>2<tab>3.2
1.9<tab>1<tab>1.0
3.2<tab>1<tab>2.0

5.1.2 Dynamic Attributes

First specify the number of static attributes used while grouping time series segments. Use the command **NUM_STATIC** to indicate the number of static attributes. Separate the command and value using a tab. For example:

NUM_STATIC 2

Then specify the number of dynamic attributes using **NUM_DYNAMIC** command. Add a tab between the command and value. For example:

NUM_DYNAMIC 3

Next, specify the sampling interval for attribute values using the **SAMPLE_INTERVAL** command. Sampling interval should be in seconds. For example:

SAMPLE_INTERVAL 300

Total number of time series samples are indicated next using the **NUM_SAMPLES** command. For example:

NUM_SAMPLES 4

To speed up the processing of the input file, we also explicitly specify the length of the longest time-series segment using the **MAX_SAMPLE_LEN** command. For example, if the longest segment has 5 samples it is specified as follows:

MAX_SAMPLE_LEN 5

Then list the attribute names using **ATTNAMES** command. First list the static attribute names and then the dynamic attribute names. Order of attributes names should be same as the order of actual data. For example:

ATTNAMES CSp NCore CFree MFree Tx

Finally, indicate the start of data using the **DATA** command. Data has the following format:

StaticAtt_Val1<tab>StaticAtt_Val2<tab>…<tab>ave_DynamicAtt1<tab>ave_DynamicAtt2<tab>…<tab>ave_DynamicAttN<tab>NoTimeSamples<tab>DynamicAtt1_Val1<tab>DynamicAtt1_Val2<tab>…<tab>DynamicAtt1_ValN<tab>DynamicAtt2_Val1<tab>DynamicAtt2_Val2<tab>…<tab>Att2ValN<tab>Att3Val1…

ave_DynamicAtti is the average of all the samples in the given segment. *NoTimeSamples* is the length of each time series segment. Consider the following example from **Dummy_Dynamic.txt** file.

DATA

3.2	2	2	1.67	6.0	3	1	2	3	1	2	2	9	7	2					
2.4	1	2.25	2.67	6.5	4	4	2	2	1	4	4	1	1.7	8	7	6	5		
2.2	2	2.8	2.8	7.0	5	5	2	2	1	4	4	1	7	1	1	9	8	5	4	9
2.8	4	2.3	4.05	2.25	4	4	2.1	2.1	1	4.1	4	1	7.1	1	2	3	3			

5.2 Queries

Queries specify the empirical CDF of number of resources requested by a query (m_q), set of states, state transitions, and their frequencies. The empirical CDF of m_q is specified after the **ECDF** command. Then each subsequent line indicates m_q and the corresponding cumulative probability separated by a tab. For example:

ECDF
1 0.41
2 0.46
3 0.48
5 0.64
6 0.73
7 0.86
8 0.96
10 1.0

Then the **STATES** command is used to indicate the state transitions. Attribute value ranges in a state (i.e., attribute) are specified as follows:

State_Min_Max

where *MIN* is the lower bound and *Max* is the upper bound. A state transition is indicated as follows:

State1_Min1_Max1<tab>State2_Min2_Max2<tab>frequency

Following examples shows a set of state transitions:

STATES
START CSp_2.0_2.8 2

CSp_2.0_2.8 FINISH 2
START CSp_2.0_3.2 1
CSp_2.0_3.2 Tx_0.0_1500.0 1
Tx_0.0_1500.0 FINISH 1
START CFree_50.0_100.0 5
CFree_50.0_100.0 FINISH 3
CFree_50.0_100.0 MSize_2.0_32.0 2
MSize_2.0_32.0 FINISH 2

5.3 Output File Formats

5.3.1 Resources

Each line in generated resources has the following format:

Node_ID<tab>Time<tab>StaticAtt_Val1<tab>StaticAtt_Val2<tab>…<tab>DynamicAtt1<tab>DynamicAtt1<tab>…

Attributes are same as the order in **Static attributes** and **Dynamic attributes** list boxes. See following example where 5,000 nodes are generated with a sampling interval of 1,800 Sec.:

1 0 2.324 8 15.674 0 10.2742 0.3 549.52
2 0 2.324 8 15.674 0 10.1282 0.1 837.27
3 0 2.324 8 14.6336 0 9.98333 0.1 752.98
4 0 2.324 8 15.674 0.3 9.0725 1.8 1250.24
5 0 2.324 8 15.674 0 10.0207 0.1 21898.8
….
4996 0 2.324 8 15.674 0 9.99 0.1 734.552
4997 0 2.324 8 15.674 0 10.09 0.1 483.62
4998 0 2.324 8 15.674 3.6 11.9245 3.4 12503.7
4999 0 2.324 8 15.674 0 10.02 0.6 430638
5000 0 2.324 24 15.674 32.8 17.29 16.4 92193.5
1 1800 2.324 8 15.674 0 10.028 5.8 400.22
2 1800 2.324 8 15.674 0 10.25 7 430.226
3 1800 2.324 8 14.6336 0 9.8425 7.8 444.58
4 1800 2.324 8 15.674 0 9.22483 1.2 7721.9
5 1800 2.324 8 15.674 0 10.31 1.3 8727.83
….
4996 1800 2.324 8 15.674 5.2 9.2 10.1 1113.42
4997 1800 2.324 8 15.674 0 10.11 0.1 413.45
4998 1800 2.324 8 15.674 11.9 9.05 0.9 915.1
4999 1800 2.324 8 15.674 0 9.7075 4.4 559.67
5000 1800 2.324 24 15.674 32.2 16.96 8.3 44398.9
1 3600 2.324 8 15.674 0 10.1433 5.8 1358.37
2 3600 2.324 8 15.674 0 10.1233 7 781.203
3 3600 2.324 8 14.6336 0 10.0773 7.7 610
4 3600 2.324 8 15.674 0 10.577 1.1 689.95
5 3600 2.324 8 15.674 0 10.01 3.6 447

5.3.2 Queries

Each query specifies the require number of resources and range of attribute values and has the following format:

m<tab>m_q<tab>AttName1<tab>Min1<tab>Max1<tab>AttName2<tab>Min2<tab>Max2…

For example:

```
m       1   Late    0.0     30.0    1mLd      0.0       2.0
m       2   DFree   100.0   400.0   MFree   1.0     4.0
m       5   Loc  10.0    10.0
m       1   1mLd    0.0   4.0
m       4   DFree   0.0   10000.0         Late    0.0         30.0    1mLd    0.0         2.0
```

6. Appendix

6.1 Compiling ResQue

MATLAB c compiler (*mcc*) is needed to compile ResQue. Download latest version of ResQue from http://www.cnrl.colostate.edu/Projects/CP2P/. Extract the folder and then set the **Current Folder** in MATLAB to where the extracted files are copied.

Step 1 In the MATLAB shell type **deploytool**. This will open up a dialog box.

Step 2 Set the project name to **ResQue.prj**. Also set the **Location** to save the project. Suppose we want to create a Windows Standalone Application. Therefore, set the application **Type** to **Windows Standalone Application**. Then click **Ok** button. This will open up the Windows Standalone Application pane.

Step 3 On the **Build** tab click on the **Add Main File Link** and then select **ResQue.m**.

Step 4 To enable running ResQue without MATLAB we need to add MCR file to the installation program. Click on the **Package** tab and click on **Add MCR** link.

Step 5 Click on the **Package** button to generate the package or if you wish to use it on the same machine you may use the **Build** button. This may take a few minutes.

You may also compile each of the .fig files using *mcc* command which has the following format:

```
mcc [-options] mfile1          [mfile2 … mfileN]
```

Example:

```
mcc –m ResQue.m
```

6.2 Error Messages

- Error in attribute names (ATTNAMES).
  - Error in list of attribute names. Check line starting with ATTNAMES
- No of attributes in the list of names & data don't match.
  - No of attributes in DTAT tag is not same as no of attributes given in ATTNAMES tag
- Unknown tag in static attribute file.
  - File has an unknown tag(s) or file is miss formatted
- Error in no of static attributes (NUM_STATIC).
  - Make sure correct no of static attributes is given
- Error in no of dynamic attributes (NUM_DYNAMIC).
  - Make sure correct no of dynamic attributes is given
- Error in sampling interval (SAMPLE_INTERVAL).
  - Make sure valid sampling interval is given
- Error in number of samples (NUM_SAMPLES).
  - Make sure correct no of time series samples is given

- Error in MAX sample length (MAX_SAMPLE_LEN).
    - Make sure the length of the longest series sample(s) is given correctly. Length is specified based on time per attribute. Not the number of columns in a line.
- No of attributes in ATTNAMES & DATA don't match.
    - Make sure number of attributes given in with ATTNAMES command is same as what is given under DATA.
- No of samples in time series not correct.
    - No of columns in each line under DATA should be equal to NUM_STATIC + 1 + numTSSamples * NUM_DYNAMIC. numTSSamples is the values specified in column NUM_STATIC + 1.
- No of time series samples != #
    - Actual no of samples given under DATA is not equal to NUM_SAMPLES
- Cannot correlate with more than 1 static attribute
    - Correlation between static and dynamic attributes can be established using only 1 attribute. Multiple attributes can be used by modifying the source code.
- Static attribute selected for correlation not in time-series library
    - Static attribute selected to establish correlation with dynamic attributes in in time series library file. Make sure static and dynamic attribute datasets belong to the same dataset such as PlanetLab.
- Static attribute selected to establish correlation must be selected as one of the static attributes to include in generated data.
    - Attributes to generate (once that will be saved in output file) must include the attribute selected to establish correlation with dynamic attributes.
- Error in ECDF data. No of columns != 2.
    - Error in empirical CDF data for number of resources specified in a query. Accepted format: <no_resources><tab><CDF value>
- Error in STATES data. No of columns != 3.
    - Error in state transition data. Accepted format: <state1><tab><state2><tab><no_of_transitions>
- Unknown tag in query state file.
    - File has an unknown tag(s) or file is miss formatted