NAME

rabins – process **argus(8)** data within specified bins.

SYNOPSIS

rabins [−B *secs***] −M** *splitmode* **[***options***]] [raoptions] [--** *filter-expression***]**

DESCRIPTION

Rabins reads **argus** data from an *argus-data* source, and adjusts the data so that it is aligned to a set of bins, or slots, that are based on either time, input size, or count. The resulting output is split, modified, and optionally aggregated so that the data fits to the constraints of the specified bins. **rabins** is designed to be a combination of **rasplit** and **racluster**, acting on multiple contexts of argus data.

The principal function of **rabins** is to align input data to a series of bins, and then process the data within the context of each bin. This is the basis for real-time stream block processing. Time series stream block processing is cricital for flow data graphing, comparing, analyzing, and correlation. Fixed load stream block processing, based on the number of argus data records ('count'), or a fixed volume of data ('size') allows for control of resources in processing. While load based options are very useful, they are rather esoteric. See the online examples and rasplit.1 for examples of using these modes of operation.

Time Series Bins

Time series bin'ing is specified using the **-M** *time* option. Time bins are specified by the size and granularity of the time bin. The granularity, 's'econds, 'm'inutes, 'h'ours, 'd'ays, 'w'eeks, 'M'onths, and 'y'ears, dictates where the bin boundaries lie. To ensure that 0.5d and 12h start on the same point in time, second, minute, hour, and day based bins start at midnight, Jan 1st of the year of processing. Week, month and year bins all start on natural time boundaries, for the period.

rabins provides a separate processing context for each bin, so that aggregation and sorting occur only within the context of each time period. Records are placed into bins based on load or time. For load based bins, input records are processed in received order and are not modified. When using time based bins, records are placed into bins based on the starting time of the record. By default, records that span a time boundary are split into as many records as needed to fit the record into appropriate bin sizes, using the algorithms used by **rasplit.1**. Metrics are distributed uniformly within all the appropriate bins. The result is a series of data and/or fragments that are time aligned, appropriate for time seried analysis, and visualization.

When a record is split to conform to a time series bin, the resulting starting and ending timestamps may or may not coincide with the timestamps of the bins themselves. For some applications, this treatment is critical to the analytics that are working on the resulting data, such as transaction duration, and flow traffic burst behavior. However, for other analytics, like average load, and rate analysis and reporting, the timestamps need to be modified so that they reflect the time range of the actual time bin boundaries. Rabins supports the optional **hard** option to specify that timestamps should conform to bin boundaries. One of the results of this is that all durations in the reported records will be the bin duration. This is extremely important when processing certain time series metrics, like load.

Load Based Bins

Load based bin'ing is specified using the **-M size** or **-M count** options. Load bins are used to constrain the resource used in bin processing. So much load is input, aggregation is performed on the input load, and when a threshold is reached, the entire aggregation cache is dumped, reinitiallized, and reused. These can be used effectively to provide realtime data reduction, but within a fixed amount of memory.

Output Processing

rabins has two basic modes of output, the default holds all output in main memory until EOF is encountered on input, where each sorted bin is written out. The second output mode, has **rabins** writing out the contents of individual sorted bins, periodically based on a holding time, specified using the *-B secs* option.

The *secs* value should be chosen such that **rabins** will have seen all the appropriate incoming data for that time period. This is determined by the ARGUS_FLOW_STATUS_INTERVAL used by the collection of argus data sources in the input data stream, as well as any time drift that may exist amoung argus data processin elements. When there is good time sync, and with an ARGUS_FLOW_STATUS_INTERVAL of 5 seconds, appropriate *secs* values are between 5-15 seconds.

The output of **rabins** when using the *-B secs* option, is appropriate to drive a number of processing elements, such as near real-time visualizations and alarm and reporting.

Output Stream

Like all **ra.1** client programs, the output of **rabins.1** is an argus data stream, that can be written as binary data to a file or standard output, or can be printed. **rabins** supports all the output functions provided by **rasplit.1**.

The output files name consists of a prefix, which is specified using the *-w ra option*, and for all modes except **time** mode, a suffix, which is created for each resulting file. If no prefix is provided, then **rabins** will use 'x' as the default prefix. The suffix that is used is determined by the mode of operation. When **rabins** is using the default count mode or the size mode, the suffix is a group of letters 'aa', 'ab', and so on, such that concatenating the output files in sorted order by file name produces the original input file. If **rabins** will need to create more output files than are allowed by the default suffix strategy, more letters will be added, in order to accomodate the needed files.

When **rabins** is spliting based on time, **rabins** uses a default extension of %Y.%m.%d.%h.%m.%s. This default can be overrided by adding a '%' extension to the name provided using the *-w* option.

When standard out is specified, using *-w -*, **rabins** will output a single **argus-stream** with START and STOP argus management records inserted appropriately to indicate where the output is split. See **argus(8)** for more information on output stream formats.

When **rabins** is spliting on output record count (the default), the number of records is specified as an ordinal counter, the default is 1000 records. When **rabins** is spliting based on the maximum output file size, the size is specified as bytes. The scale of the bytes can be specified by appending 'b', 'k' and 'm' to the number provided.

When **rabins** is spliting base on time, the time period is specified with the option, and can be any period based in seconds (s), minutes (m), hours (h), days (d), weeks (w), months (M) or years (y). **Rabins** will create and modify records as required to split on prescribed time boundaries. If any record spans a time boundary, the record is split and the metrics are adjusted using a uniform distribution model to distribute the statistics between the two records.

See **rasplit.1** for specifics.

RABINS SPECIFIC OPTIONS

rabins, like all ra based clients, supports a number of **ra options** including remote data access, reading from multiple files and filtering of input argus records through a terminating filter expression. Rabins also provides all the functions of **racluster.1** and **rasplit.1**, for processing and outputing data. **rabins** specific options are:

−B *secs*
Holding time in seconds before closing a bin and outputing its contents.

−M *splitmode*
Supported spliting modes are:

time <n[smhdwMy]>
bin records into time slots of n size. This is used for time series analytics, especially graphing. Records, by default are split, so that their timestamps do not span the time range specified. Metrics are uniformly distributed among the resulting records.

count <n[kmb]>
bin records into chunks based on the number of records. This is used for archive management and parallel processing analytics, to limit the size of data processing to fixed numbers of records.

size <n[kmb]>
bin records into chunks based on the number of total bytes. This is used for archive management and parallel processing analytics, to limit the size of data processing to fixed byte limitations.

−M *modes*
Supported processing modes are:
hard
split on hard time boundaries. Each flow records start and stop times will be the time boundary times. The default is to use the original start and stop timestamps from the records that make up the resulting aggregation.
nomodify
Do not split the record when including it into a time bin. This allows a time bin to represent times outside of its defintion. This option should not be used with the 'hard' option, as you will modify metrics and semantics.

−m *aggregation object*
Supported aggregation objects are:

none	use a null flow key.
srcid	argus source identifier.
smac	source mac(ether) addr.
dmac	destination mac(ether) addr.
soui	oui portion of the source mac(ether) addr.
doui	oui portion of the destination mac(ether) addr.
smpls	source mpls label.
dmpls	destination label addr.
svlan	source vlan label.
dvlan	destination vlan addr.
saddr/[l|m]	source IP addr/[cidr len | m.a.s.k].
daddr/[l|m]	destination IP addr/[cidr len | m.a.s.k].
matrix/l	sorted src and dst IP addr/cidr len.
proto	transaction protocol.
sport	source port number. Implies use of 'proto'.
dport	destination port number. Implies use of 'proto'.
stos	source TOS byte value.
dtos	destination TOS byte value.
sttl	src -> dst TTL value.
dttl	dst -> src TTL value.
stcpb	src -> dst TCP base sequence number.
dtcpb	dst -> src TCP base sequence number.
inode[/l|m]]	intermediate node IP addr/[cidr len | m.a.s.k], source of ICMP mapped events.

sco	source ARIN country code, if present.
dco	destination ARIN country code, if present.
sas	source node origin AS number, if available.
das	destination node origin AS number, if available.
ias	intermediate node origin AS number, if available.

–P *sort field*

Rabins can sort its output based on a sort field specification. Because the **-m** option is used for aggregation fields, **-P** is used to specify the print priority order. See **rasort(1)** for the list of sortable fields.

–w *filename*

Rabins supports an extended *-w* option that allows for output record contents to be inserted into the output filename. Specified using '$' (dollar) notation, any printable field can be used. Care should be taken to honor any shell escape requirements when specifying on the command line. See **ra(1)** for the list of printable fields.

Another extended feature, when using **time** mode, **rabins** will process the supplied filename using **strftime(3)**, so that time fields can be inserted into the resulting output filename.

INVOCATION

This invocation aggregates **inputfile** based on 10 minute time boundaries. Input is split to fit within a 10 minute time boundary, and within those boundaries, argus records are aggregated. The resulting output its streamed to a single file.

rabins -r * -M time 10m -w outputfile

This next invocation aggregates **inputfiles** based on 5 minute time boundaries, and the output is written to 5 minute files. Input is split such that all records conform to hard 10 minute time boundaries, and within those boundaries, argus records are aggregated, in this case, based on IP address matrix.
The resulting output its streamed to files that are named relative to the records output content, a prefix of */matrix/%Y/%m/%d/argus.* and the suffixes *%H.%M.%S*.

rabins `-r * -M hard time 5m -m matrix -w "/matrix/%Y/%m/%d/argus.%H.%M.%S"`

This next invocation aggregates **input.stream** based on matrix/24 into 10 second time boundaries, holds the data for an additional 5 seconds after the time boundary has passed, and then prints the complete sorted contents of each bin to standard output. The output is printed at 10 second intervals, and the output is the content of the previous 10 sec time bin. This example is meant to provide, every 10 seconds, the summary of all Class C subnet activity seen. It is intended to run indefinately printing out aggregated summary records. By modifying the aggregation model, using the "-f racluster.conf" option, you can achieve a great deal of data reduction with a lot of semantic reporting.

```
% rabins -S localhost -m matrix/24 -B 5s -M hard time 10s -p0 -s +1trans - ipv4
           StartTime  Trans  Proto            SrcAddr   Dir            DstAddr  SrcPkts  DstPkts     SrcBytes     DstBytes State
 2012/02/15.13:37:00      5     ip     192.168.0.0/24   <->     192.168.0.0/24       41       40         2860        12122   CON
 2012/02/15.13:37:00      2     ip     192.168.0.0/24    ->       224.0.0.0/24        2        0          319            0   INT
[ 10 seconds pass]
 2012/02/15.13:37:10     13     ip     192.168.0.0/24   <->    208.59.201.0/24      269      351        97886       398700   CON
 2012/02/15.13:37:10     14     ip     192.168.0.0/24   <->     192.168.0.0/24       86       92         7814        46800   CON
 2012/02/15.13:37:10      1     ip    17.172.224.0/24   <->     192.168.0.0/24       52       37        68125         4372   CON
 2012/02/15.13:37:10      1     ip     192.168.0.0/24   <->      199.7.55.0/24        7        7          784         2566   CON
 2012/02/15.13:37:10      1     ip     184.85.13.0/24   <->     192.168.0.0/24        6        5         3952         2204   CON
 2012/02/15.13:37:10      2     ip    66.235.132.0/24   <->     192.168.0.0/24        5        6          915         3732   CON
 2012/02/15.13:37:10      1     ip    74.125.226.0/24   <->     192.168.0.0/24        3        4          709          888   CON
 2012/02/15.13:37:10      3     ip       66.39.3.0/24   <->     192.168.0.0/24        3        3          369          198   CON
 2012/02/15.13:37:10      1     ip     192.168.0.0/24   <->     205.188.1.0/24        1        1           54          356   CON
[ 10 seconds pass]
 2012/02/15.13:37:20      6     ip     192.168.0.0/24   <->    208.59.201.0/24      392      461        60531       623894   CON
 2012/02/15.13:37:20      8     ip     192.168.0.0/24   <->     192.168.0.0/24       95      111         6948        93536   CON
 2012/02/15.13:37:20      3     ip     72.14.204.0/24   <->     192.168.0.0/24       38       32        38568         4414   CON
```

```
 2012/02/15.13:37:20      1     ip    17.112.156.0/24   <->     192.168.0.0/24       26       13        21798         7116   CON
 2012/02/15.13:37:20      2     ip    66.235.132.0/24   <->     192.168.0.0/24        6        3         1232         4450   CON
 2012/02/15.13:37:20      1     ip    66.235.133.0/24   <->     192.168.0.0/24        1        2           82          132   CON
[ 10 seconds pass]
 2012/02/15.13:37:30    117     ip     192.168.0.0/24   <->    208.59.201.0/24      697      663       369769       134382   CON
 2012/02/15.13:37:30     11     ip     192.168.0.0/24   <->     192.168.0.0/24      147      187        11210       193253   CON
 2012/02/15.13:37:30      1     ip     184.85.13.0/24   <->     192.168.0.0/24       13        9        13408         9031   CON
 2012/02/15.13:37:30      2     ip    66.235.132.0/24   <->     192.168.0.0/24        8        7         1920        11563   CON
 2012/02/15.13:37:30      1     ip     192.168.0.0/24   <->    207.46.193.0/24        5        3          802          562   CON
 2012/02/15.13:37:30      1     ip    17.112.156.0/24   <->     192.168.0.0/24        5        2          646         3684   CON
 2012/02/15.13:37:30      2     ip     192.168.0.0/24    ->       224.0.0.0/24        2        0          382            0   REQ
[ 10 seconds pass]
```

This next invocation reads IP argus(8) data from inputfile and processes, the argus(8) data stream based on input byte size of no greater than 1 Megabyte. The resulting output stream is written to a single *argus.out* **data file.**

rabins -r argusfile -M size 1m -s +1dur -m proto -w argus.out - ip

This invocation reads IP argus(8) data from inputfile and aggregates the argus(8) data stream based on input file size of no greater than 1K flows. The resulting output stream is printed to the screen as standard argus records.

rabins -r argusfile -M count 1k -m proto -s stime dur proto spkts dpkts - ip

COPYRIGHT

Copyright (c) 2000-2014 QoSient. All rights reserved.

SEE ALSO

ra(1), racluster(1), rasplit(1), rarc(5), argus(8),

AUTHORS

Carter Bullard (carter@qosient.com).