Cisco Crosswork Situation Manager 7.2.x Clustering Algorithm Guide

Powered by Moogsoft AIOps 7.2

Sigalisers are the clustering algorithms in Cisco Crosswork Situation Manager that group alerts based on factors such as time, language, similarity and proximity.

The clustering algorithms available include:

- Cookbook
- Tempus
- Feedback

You can configure and run multiple different clustering algorithms on the same instance of Cisco Crosswork Situation Manager. The algorithms you choose depend on your specific use case and the type of Situations you want your operators to receive.

You can also apply entropy and Vertex Entropy calculations to add another degree of filtering to the alerts you want to correlate. For example, you can use an entropy threshold if you want to exclude alerts with low operational value or include alerts with high operational value. See Vertex Entropy and Entropy for more details.

Cookbook

Cookbook is a clustering algorithm that creates clusters defined by the relationships between alerts and their attributes.

Type: Attribute-based clustering.

Use case: You can use Cookbook if you want more control in how you correlate alerts based on patterns in the text similarity. Example use cases include:

- Grouping alerts with a similar description and from the same application or service
- Grouping alerts from the same host or location.
- Topology-based correlation using Vertex Entropy.

Benefits: Cookbook offers the following advantages:

- Very customizable and configurable using Recipes.
- Able to create Situations when an alert exceeds a defined rate of occurrence.
- Can include and exclude alerts that meet specific criteria such as Vertex Entropy.
- Able to partition alerts into Situations using textual similarity-based comparison.
- Possible to base alert clustering on topological relationships.

Note: Cookbook can be difficult to manage if you implement at scale if you have large or complex rule sets.

Configuration: Both UI and backend configuration. See Cookbook for details.

Tempus

Tempus is a time-based algorithm that clusters alerts into Situations based on the similarity of their timestamps.

Type: Time-based clustering.

Use case: You want to match alerts based on patterns in their timestamps or on a timeline. Use Tempus if you want your alerts to be clustered in real-time. The logic behind Tempus is that a triggering event causes additional subsequent failures within a short timeframe. Works well in scenarios where there is a causal chain such as:

- Cascading failures
- Performance failures
- Brownouts

Benefits: Tempus offers the following advantages:

- No enrichment required. See Enrichment.
- Good for availability alerts.
- Good for performance alerts.

Note: Tempus is limited to only matching alerts by time proximity and patterns so it can be difficult to know the full context of the Situations it creates. For example, you might receive alerts from unrelated failures if they happen at the same time. Tempus also has an inherent degree of fuzziness or wrongness.

Configuration: Backend configuration only. See Tempus for details.

Feedback

Warning: Feedback is a Beta feature.

Feedback is the neural-based algorithm that learns and unlearns actions based on user feedback.

Type: Neural/learns user feedback.

Use case: Feedback is currently a prototype and should not be used in production environments. You can use it if the other clustering algorithms did not correlate anything, as you can teach it what to cluster. For example, if you have a set of alerts that you want to cluster but they didn't cluster through time, attribute similarity or topological proximity, you can teach the system and it learns to cluster those alerts.

Alternatively, you might want to use Feedback if you want to manually create Situations and teach Cisco Crosswork Situation Manager to cluster the same type of alerts. Another use case is to use Feedback alongside Tempus. If you have several team members looking at time-based correlation with an inherent degree of fuzziness, they can use Feedback to train the system to remember good Situations and forgot about bad Situations and persist that behavior in future. For example, you could teach it to remember when there was a server failure but to ignore the printer ink failure and persist that behavior.

Benefits: Feedback offers the following advantages:

- No enrichment required. See Enrichment.
- Allows operators to push domain knowledge back into the system.
- Can be trained to only create the Situations you are interested in.

Note: If you train Feedback badly, it clusters badly.

Configuration: Both UI and backend configuration. See Feedback for more details.

Cookbook

Cookbook is a deterministic clustering algorithm in Cisco Crosswork Situation Manager that creates Situations defined by the relationships between alerts.

You can configure Cookbook to cluster alerts into Situations if they have specific characteristics such as temporal or topological proximity. Cookbook filters can include characteristics such as the following:

- Class or type
- Description
- Server priority
- Geographical location
- Environment classification

Each Cookbook is a collection of Recipes: sets of configurable filters, triggers, and other calculations such as priority ordering and entropy threshold. A Cookbook can run multiple Recipes concurrently to process the incoming event stream and produce a variety of Situations. A Cisco Crosswork Situation Manager deployment may include multiple instances of Moogfarmd, each of which can run multiple Cookbooks.

To configure a Cookbook and its recipes via the Cisco Crosswork Situation Manager UI, see Configure a Cookbook.

To use more advanced features, such as merging and Moobot-controlled Recipes in moogfarmd, see Configure a Cookbook Manually.

Cookbooks configured in the UI and in Moogfarmd can run concurrently.

Configure a Cookbook Manually

Cookbook is a deterministic clustering algorithm in Cisco Crosswork Situation Manager that creates Situations defined by the relationships between alerts.

You can install a basic Cookbook via the UI by supplying a name, the Recipes you want it to use and configuring a few of the available properties. See Configure a Cookbook for setup steps.

Configure an advanced Cookbook if you want to configure additional properties such as determining if it runs on startup, selecting a different Moobot and other options available not available in the UI.

Before You Begin

Before you set up your Cookbook, ensure you have met the following requirements:

- You have set up the recipes you want your Cookbook to use. See Configure a Cookbook Recipe for details.
- Your LAMs or integrations are running and Cisco Crosswork Situation Manager is receiving events.
- You have configured the Moolet that you want Cookbook to process the output of.

Configure an Advanced Cookbook

Edit the configuration file at **`$MOOGSOFT_HOME/config/moolets/cookbook.conf`** to control the behavior of the Cookbook.

See the Cookbook and Recipe Reference for a full description of all properties. Some properties in the file are commented out by default. Uncomment the properties to enable them.

1. Provide a name and description for the Cookbook.
   - name:Name of the Cookbook (required).
   - classname: Class of the Moolet. Do not change.
   - description:Text description of the Cookbook.
2. Configure the Cookbook's behavior for when it starts and stops running:
   - run_on_startup: Determines whether Cookbook runs when Cisco Crosswork Situation Manager starts.
   - metric_path_moolet: Determines whether or not Cisco Crosswork Situation Manager includes Cookbook in the Event Processing metric for Self Monitoring.
   - moobot: Defines the Moobot that Cookbook loads at startup.
   - process_output_of: Defines the Moolet sources of the alerts that Cookbook processes.
3. Configure the Cookbook algorithm and how it clusters alerts:
   - membership_limit: Maximum number of Situations that an alert can be a member of.
   - scale_by_severity: Cookbook treat alerts with a high severity like alerts with a high entropy value.
   - entropy_threshold: Minimum entropy value an alert must have in order for Cookbook to include it in a Situation.
   - single_recipe_matching: Enables Cookbook to treat Recipes in priority order.
   - cluster_match_type: Defines the Cookbook cluster matching method.
   - cook_for: Minimum time period that Cookbook clusters alerts for before the Recipe resets.
   - cook_for_extension: Time period that Cookbook can extend clustering alerts for before the Recipe resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Cookbook.
   - max_cook_for: Maximum time period that Cookbook clusters alerts for before the Recipe resets and starts a new cluster.
4. Select and name the Recipe(s) you want Cookbook to use:
   - chef: Type of recipe you want to use. CValueRecipeV2, CValueRecipe or CBotRecipe.
   - name: Name of the Recipe.
   - description: Description of the Recipe.
5. Configure the Recipe behavior and the filters that define the alert relationships. See Configure a Cookbook Recipe for more details.
   - recipe_alert_threshold: Maximum number of alerts to cluster before Cookbook creates a Situation.
   - exclusion: Filter determining the alerts to exclude from Situation creation.
   - trigger: Filter determining the alerts that Cookbook considers for Situation creation.
   - seed_alert: Filter determining whether to create a Situation from a seed alert.

- rate: Filter determining the minimum event rate per minute required for Cookbook to create a Situation.
- min_sample_size: Minimum number of events contained in a cluster before Cisco Crosswork Situation Manager calculates the rate.
- max_sample_size: Maximum number of events contained in a cluster before Cisco Crosswork Situation Manager calculates the rate.
- cook_for: Minimum time period that Cookbook clusters alerts for before the recipe resets. The Cookbook cook_for value overwrites this if it exists.
- cook_for_extension: Time period that Cookbook can extend clustering alerts for before the Recipe resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Recipe.
- max_cook_for: Maximum time period that Cookbook clusters alerts for before the Recipe resets and starts a new cluster.

6. Configure the alert matching properties for the Recipe:
   - cluster_match_type: Defines how Cookbook matches alerts to clusters.
   - hop_limit: Maximum number of hops between the alert source nodes in order for the alerts to qualify for clustering.
   - components: Values that alerts must match for Cookbook to include them in a Situation.

Restart the Moogfarmd service to activate any changes you make to the configuration file. See Control Moogsoft AIOps Processes for further details.

Example

The following example demonstrates a Cookbook that uses a CValueRecipeV2 that splits alerts into clusters with both a **source_id** (hostname) and a **description** that are 50% similar. You control this under the components property:

```
matcher:{
        components:[
        {
                name:"source_id",
                similarity:0.5,
        },
        {
                name:"description",
                similarity:0.5,
        }
}
```

It also only creates a Situation from a seed alert with a Vertex Entropy value of 0.75 which indicates a node of high topological importance. You control the seed alert filter with this property:

```
 seed_alert:"vertex_entropy = 0.75",
```

The full example Cookbook with both configurations is as follows:

```
{
   name:"Cookbook",
   classname:"CCookbook",
   run_on_startup:true,
```

```
    persist_state:false,
    metric_path_moolet:true,
    moobot:"Cookbook.js",
    process_output_of:"MaintenanceWindowManager",
    membership_limit:1,
    scale_by_severity:false,
    entropy_threshold:0.0,
    single_recipe_matching:false,
    cluster_match_type:"first_match",
    recipes:[
       {
          chef:"CValueRecipeV2",
          name:"SplitBySourceAndDescription",
          description:"Value Recipe outage",
          recipe_alert_threshold:0,
          exclusion:"severity < 5",
          trigger:null,
          seed_alert:"vertex_entropy = 0.75",
          rate:0,
          min_sample_size:5,
          max_sample_size:10,
          #cook_for:5000,
          cluster_match_type:"first_match",
          matcher:{
             hop_limit:2,
             components:[
                {
                   name:"source_id",
                   similarity:0.5,
                   shingle_size:4
                },
                {
                   name:"description",
                   similarity:0.5,
                   shingle_size:-1
                }
             ]
          }
        }
       ],
       cook_for:2000
    }
```

Start or Stop the Cookbook

You can start or stop Cookbook if you want to make additional changes. You must always restart Moogfarmd for these changes to take effect.

Note that restarting Moogfarmd clears any existing clusters from Cisco Crosswork Situation Manager.

See Control Moogsoft AIOps Processes for further details.Control Moogsoft AIOps Processes

Configure a Recipe Manually

A Cookbook Recipe is a set of configurable filters, triggers, and calculations that defines the type of alerts and the alert relationships that Cookbook detects and clusters into Situations.

When you add Recipes from moogfarmd, you can only configure advanced properties such complex Cookbook such as calling Moobot functions. You can configure three recipe types: CValueRecipeV2, CValueRecipe and CBotRecipe. See Recipe Types for more details.

Refer to Cookbook and Recipe Reference to see the available properties.

Before You Begin

Before you set up your Recipe, ensure you have met the following requirements:

- Your LAMs or integrations are running and Cisco Crosswork Situation Manager is receiving events.
- If you want to use Vertex Entropy or hop limit in your Recipes, you have imported your network topology. See Import a Topology.

Create a Cookbook Recipe

Edit the configuration file at **`$MOOGSOFT_HOME/config/moolets/cookbook.conf`** to add a new Recipe or edit an existing one.

See the Cookbook and Recipe Reference for a full description of all properties. Some properties in the file are commented out by default. Uncomment the properties to enable them.

1. Provide a name and description for the Recipe:
   - chef: Type of Recipe. Defaults to CValueRecipeV2.
   - name:Name of the Recipe.
   - description:Text description of the Recipe that appears in each Situation.
2. Configure the Recipe behavior and filters that define the alert relationships:
   - recipe_alert_threshold: Maximum number of alerts to cluster before Cookbook creates a Situation.
   - exclusion: Filter that determines the alerts to exclude from Situation creation.
   - trigger:Filter that determines the alerts that Cookbook considers for Situation creation.
   - seed_alert: Filter that determines whether to create a Situation from a seed alert.
   - rate: Filter that determines the minimum event rate per minute required for Cookbook to create a Situation.
   - min_sample_size: Minimum number of events contained in a cluster before Cisco Crosswork Situation Manager calculates the rate.
   - max_sample_size: Maximum number of events contained in a cluster before Cisco Crosswork Situation Manager calculates the rate.
   - cook_for: Minimium time period that Cookbook clusters alerts for before the recipe resets. The Recipe **`cook_for`** value overwrites the Cookbook **`cook_for`** value.

- cook_for_extension: Time period that Cookbook can extend clustering alerts for before the Recipe resets and starts a new cluster. The Recipe **cook_for_extension** value overwrites the Cookbook **cook_for_extension** value. Setting this value enables the cook for auto-extension feature for this Recipe.
- max_cook_for: Maximum time period that Cookbook clusters alerts for before the Recipe resets and starts a new cluster. The Recipe **max_cook_for** value overwrites the Cookbook **max_cook_for** value. If not set, it defaults to three times the **cook_for** value.

3. Configure the alert matching properties for the Recipe:
   - cluster_match_type: Defines how Cookbook matches alerts to clusters.
   - hop_limit: Maximum number of hops between the alert source nodes in order for the alerts to qualify for clustering.
   - components: Define additional configuration such as case sensitivity for CValueRecipe and shingle size for CValueRecipeV2.

Restart the Moogfarmd service to activate any changes you make to the configuration file. See Control Moogsoft AIOps Processes for further details.

Examples

See Recipe Examples for example configurations of different Value and Bot Recipes.

Recipe Examples

The following examples show how you can configure the different types of Recipe in the Cookbook clustering algorithm.

See Recipe Types for more details on the different Recipes available.

CValueRecipeV2 Example

The following example shows a Value Recipe V2 that clusters alerts with:

- Alert source IDs that are 75% similar when breaking the source ID into shingles of four characters.
- Alert descriptions that are 75% similar.

A shingle value of -1 or less means the Recipe compares the text similarity of entire words, rather than breaking the text into shingles. See Recipe Types for more details about the calculation.

```
{
    chef: "CValueRecipeV2",
    name: "SplitBySourceAndDescription",
    description: "Value Recipe outage",
    recipe_alert_threshold: 0,
    exclusion: "severity < 5",
    trigger: "severity > 3",
    seed_alert: "vertex_entropy > 0.8",
    rate: 0,
    #Given in events per minute
    min_sample_size: 5,
    max_sample_size: 10,
    cook_for: 5000,
    cluster_match_type : "first_match",
    matcher : {
```

```
        components: [
              { name: "source_id",   similarity: 0.75, shingle_size: 4 },
              { name: "description", similarity: 0.75, shingle_size: -1 }
                        ]
       }
 },
```

CValueRecipe Example

The following Value Recipe example shows a Recipe that splits alerts into clusters with either an identical **source_id** (hostname) or a **description** that is 50% similar. It also only creates a Situation from a seed alert with a Vertex Entropy value of 0.75, which indicates a node of high topological importance. See Vertex Entropy for more information.

```
{
    chef: "CValueRecipe",
    name: "SplitBySourceAndDescription",
    description: "Value Recipe outage",
    recipe_alert_threshold: 0,
    exclusion: "severity < 5",
    trigger: null,
    seed_alert: "vertex_entropy = 0.75",
    rate: 0,
    #Given in events per minute
    min_sample_size: 5,
    max_sample_size: 10,
    cook_for: 5000,
    cluster_match_type : "first_match",
    matcher : {
        components: [
              { name: "source_id",   similarity: 1.0, case_sensitive: true },
              { name: "description", similarity: 0.5, case_sensitive: true }
                        ]
       }
 },
```

The following CValueRecipe example shows a Recipe that can be used alongside a New Relic integration.

This recipe clusters alerts that have an identical **source_id** (hostname) every fifteen minutes:

```
{
    chef: "CValueRecipe",
    name: "New Relic Hostname Recipe",
    description: "Recipe to create situations based on 100% similarity of the
hostname received from New Relic",
    recipe_alert_threshold: 1,
    exclusion: null,
    trigger: null,
    seed_alert: null,
    rate: 0,
    #Given in events per minute
    min_sample_size: 5,
    max_sample_size: 10,
    cook_for: 900,
```

```
    matcher: {
       components: [
              { name: "source_id",   similarity: 1.0, case_sensitive: true },
                ]
        }
},
```

CBotRecipe Example

The example Bot Recipe below shows a recipe that uses methods in the Cookbook.js Moobot to cluster by topological similarity.

It excludes alerts that have a severity of less than minor and clusters alerts that are 80% similar.

```
{
    chef: "CBotRecipe",
    name: "MaxwellDaemon",
    description: "Maxwell Recipe outage",
    recipe_alert_threshold: 0,
    trigger: null,
    exclusion: "severity < 3",
    rate: 1,
    #Given in events per minute
    min_sample_size: 5,
    max_sample_size: 10,
    cluster_match_type : "first_match",
    matcher: {
        initialise_function: "initBuckets",
        member_function: "checkBucket",
        similarity: 0.8
    }
    cook for: 2000,
}
```

Recipe Types

The Cookbook sigaliser uses the following Recipe types to define alert relationships and control how the Sigaliser clusters alerts:

- CValueRecipeV2
- CValueRecipe
- CBotRecipe

The CValueRecipeV2 and CValueRecipe use different methods to calculate the textual similarity between alerts. The CBotRecipe is a customizable recipe that allows you to call specific functions from a Moobot.

CValueRecipeV2

CValueRecipeV2 extracts and analyzes groups of consecutive characters to measure text similarity between alerts. It is the default Recipe in Cookbook for new Cisco Crosswork Situation Manager v7 installations and for any new Cookbooks you create.

This recipe uses the bag-of-words model and shingling natural language processing methods to calculate the text similarity between alerts. Shingling is the process in which Cookbook extracts groups of consecutive characters called shingles from a source string. Potential sources include the alert source ID or description. To measure

similarity, Cookbook calculates the number of identical shingles. You can control the calculation using the **shingle_size** component property.

For example, if you set the **shingle_size** to 2 and Cookbook receives two alerts with the source IDs:

webserver0100
webserver0200
Cookbook extracts the following shingles from the source ID strings:

we eb bs se er rv ve er r0 01 10 00
we eb bs se er rv ve er r0 02 20 00
Ten out of the 12 shingles are identical which indicates a high similarity.

If you set the **shingle_size** to 0 or less, Cookbook treats the string values as words in its text similarity calculation. In the UI, you select whether Cookbook treats string values as shingles or words.

For example, if Cookbook receives two alerts with the source IDs: "database01" and "database02", it treats them as:

database01
database02
These two words are not identical so the two alerts would be given a low similarity.

The default shingle size settings in the CValueRecipeV2 are optimal for most use cases.

CValueRecipe

The first version of the CValueRecipe that uses a string comparison mechanism to cluster alerts by textual similarity.

CValueRecipe uses string metric algorithms to calculate similarity. The calculation breaks strings up into partitions and performs a character-by-character comparison of each partition to measure similarity.

The following example sets the Recipe to monitor alerts with source IDs and descriptions with a similarity of 1.0 on a scale of 0 to 1 where 0 is dissimilar and 1 is identical:

```
matcher : {
       components: [
       { name: "source_id",   similarity: 1.0, case_sensitive: true },
       { name: "description", similarity: 1.0, case_sensitive: true }
                   ]
       }
```

In a scenario where Cookbookreceives the following alerts:

Alert	source_id	description
A	001	database
B	001	webserver
C	002	database
D	002	database

Based on the Recipe configuration, Cookbook creates three clusters: one containing alert A, one containing alert B and one containing alerts C and D which had identical source IDs and descriptions. The string may contain non-alphabetical characters. CValueRecipe can also convert numeric values to strings for comparison.

The Value Recipe uses the **case_sensitive** component to enable or disable case sensitivity as a factor in text similarity matching. For example, if you want source IDs to only match if case sensitivity is identical but you do not want descriptions to be case sensitive:

```
matcher: {
           components: [
           { name: "source_id", similarity: 1.0, case_sensitive: true}
           { name: "description", similarity: 0.7, case_sensitive: false}
                          ]
           }
```

If you do not enable case sensitivity, then an alert from a source called "WebServer1" and an alert from a source called "webserver1" would have a lesser similarity.

To make Cookbook match each value in the list individually, set "treat_as : list" in the component configuration. For example: **components: [{ name: "custom_info.source_id", similarity: 1.0, treat_as: "list" }]**. If you do not use this configuration, Cookbook treats the **components** value as a string.

CBotRecipe

CBot Recipe is a customizable Recipe that allows you to call certain functions from the Cookbook.js Moobot.

You can configure the Bot recipe to call functions defined in the Cookbook.js Moolet. The Cookbook Moolet defines two functions, an initialisation function called**initialise_function** and a **member_function**. You can call the **initialise_function** once to set up any necessary initialisation of the algorithms you want to write in the Moobot:

```
matcher :
                  {
                  initialise_function : "initBuckets",
                  member_function : "checkBucket",
                  similarity : 0.8
                  }
```

You can call the **member_function** once for every event that passes the trigger. Cookbook considers each of these events for matching and for every candidate cluster in the system.

For example, Cookbook calls the **member_function** 100 times if there are 100 candidate clusters for each alert that comes through the system. Cookbook compares the alert to candidate clusters that are potential Situations. If the alert's similarity matches or exceeds the matcher value, Cookbook adds the alert to the candidate cluster.

Match List Items in Recipes

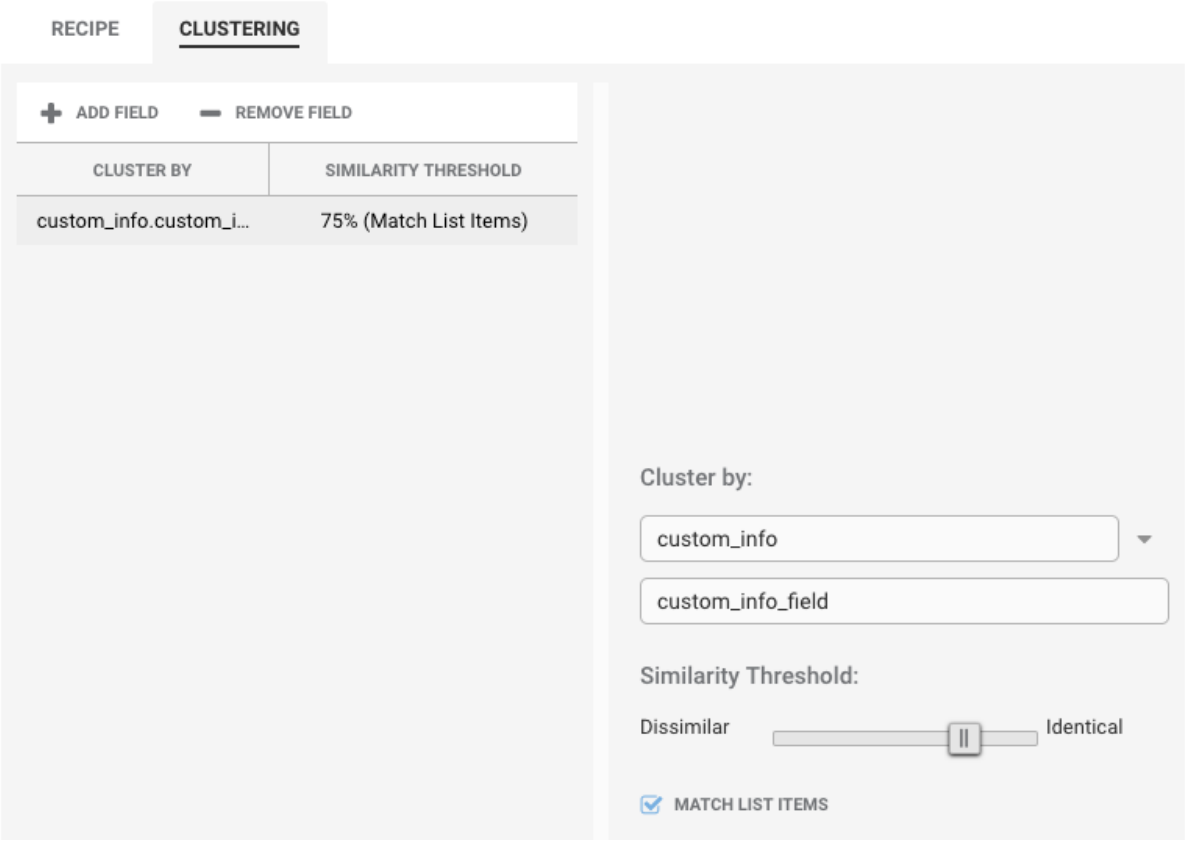
You can create Recipes and configure clustering around the use of 'custom_info' list-based fields in Alert Custom Info.

You can also set whether list-based clustering of a custom_field is applied. If not, the field will be treated as string.

Match List Items for a Custom Info Field

To match list items for a custom info field:

1. Click on the Clustering tab.
2. Select the 'custom_info' attribute from the Cluster By list. Enter the custom_info field name in the box below.
3. Check the box next to Match List Items to match individual items in custom_info lists.



Configure List-based Matching

You can also set list-based matching for Cookbook Recipes defined in **`cookbook.conf`**.

To enable this:

1. Edit **`$MOOGSOFT_HOME/config/moolets/cookbook.conf`**. See Cookbook and Recipe Reference for all available Recipe properties.
2. Add a qualifier **`treat_as: "list"`** for any custom_info components in the matcher:

```
matcher : {
            components: [ { name: "custom_info.cities", similarity: 0.5, treat_as:
 "list" } ]
          }
```

3. Save any changes and restart moogfarmd.

After configuring the Recipe, you can expect the following alerts to arrive in your system:

```
Alert 1: custom_info.offices = ["London"]
   Alert 2: custom_info.offices = ["London", "San Francisco", "Venice", "Bangalore"
   ]
   Alert 3: custom_info.offices = ["Venice", "Bangalore"]
   Alert 4: custom_info.offices = ["Bangalore"]
```

Example

If you configure your Recipe to treat the custom_field value as list and set the similarity to 1.0:

```
matcher : {
                components: [ { name: "custom_info.cities", similarity: 1.0, treat_
   as: "list" } ]
            }
```

This configuration would produce four clusters:

- Cluster A: Alert 1 and alert 2 match for "London".
- Cluster B: Alert 2 matches for "San Francisco".
- Cluster C: Alert 3 and alert 4 match on "Venice".
- Cluster D: Alerts 2, 3 and 4 match on "Bangalore".

This can produce four separate Situations as per the four clusters above, or two Situations because cluster D contains all the alerts in clusters B and C.

If the Recipe does not see custom_info field as a list then it treats the field as a single string. This means in this example all four alerts would end up in separate Situations with no clustering.

Cookbook and Recipe Reference

This is a reference for the Cookbook Sigaliser algorithm and its associated Recipes. The Cookbook configuration properties are found in **$MOOGSOFT_HOME/config/moolets/cookbook.conf.**

Moolet

name

Name of the Cookbook Sigaliser algorithm. Do not change.

Type: String

Required: Yes

Default: "Cookbook"

class

Moolet class name. Do not change.

Type: String

Required: Yes

Default: **"CCookbook"**

run_on_startup

Determines whether Cookbook runs when Cisco Crosswork Situation Manager starts. If you enable this property, Cookbook captures all alerts from the moment the system starts, without you having to configure or start it manually.

Type: Boolean

Required: No

Default: **false**

metric_path_moolet

Determines whether Cisco Crosswork Situation Manager includes Cookbook in the Event Processing metric for Self Monitoring.

Type: Boolean

Required: No

Default: **true**

moobot

Specifies which associated Moobot the Cookbook Moolet loads at startup.

Type: String

Required: Yes

Default: "**Cookbook.js**"

process_output_of

Defines the Moolet source of the alerts for Cookbook.

Type: List

Required: Yes

One of: **AlertBuilder**, **AlertRulesEngine**, **MaintenanceWindowManager**, **EmptyMoolet**

Default: "**MaintenanceWindowManager**"

Algorithm

membership_limit

Maximum number of Situations an alert can be part of. This does not impact alerts in merged Situations. Smaller limits result in fewer Situations with many alerts and many Situations with fewer associated alerts. Larger limits result in many Situations with few alerts and a few Situations with many alerts. The optimal value is between 1 and 5.

Type: Integer

Required: Yes

Default: **1**

scale_by_severity

Cookbook treat alerts with a high severity like alerts with a high entropy value. Cisco Crosswork Situation Manager divides the severity number by the maximum severity (5) to calculate the scale. For example, for an alert with minor severity, the entropy would be 3/5.

Type: Boolean

Required: No

Default: **`False`**

entropy_threshold

Minimum entropy value that an alert must have for Cookbook to consider it for clustering into a Situation. Cookbook does not include any alerts with an entropy value below the threshold in Situations. Set to a value between 0.0 and 1.0. The default of 0.0 means Cookbook processes all alerts.

Type: Decimal

Required: No

Default: **`0.0`**

single_recipe_matching

Enable **`single_recipe_matching`** for Cookbook to treat Recipes in priority order, based on the order of configuration in **`cookbook.conf`**. The first recipe in the list takes highest priority. If an alert appears in a Situation that a recipe with a low priority order creates, it may reappear in a Situation that a Recipe with a higher priority creates.

Type: Boolean

Required: No

Default: **`false`**

cluster_match_type

Defines how Cookbook matches clusters. You can select the **`first_match`** in order so Cookbook adds alerts to the first cluster over the similarity threshold value. This is the default behavior for Cookbook. Alternatively, select **`closest_match`**to add alerts to the cluster with the highest similarity greater than the similarity threshold value. This option may be less efficient because Cookbook needs to compare alerts against each cluster in a Recipe. The Recipe-level match type configuration overrides the Cookbook-level definition.

Type: List

Required: No

One of: **`first_match, closest_match`**

Default: **`"first_match"`**

cook_for

Minimum time period, in seconds, that Cookbook clusters alerts for before the Recipe resets and determines when to start a new cluster. You can set a different **`cook_for`** time for a Recipe, and this overrides the Cookbook value. Recipes without **`cook_for`** values inherit the value from the Cookbook.

Type: Integer

Required: No

Default: **`"5000"`**

cook_for_extension

Time period, in seconds, that Cookbook can extend clustering alerts for before the Recipe resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Cookbook. As Cookbook receives

related alerts, it continues to extend the total clustering time until the **max_cook_for** period is reached. Used in conjunction with the **max_cook_for** value, the **cook_for_extension** helps to ensure that Cookbook continues to cluster alerts together that are related to the same failure. The **cook_for_extension** only applies to new related alerts; it does not apply to existing alerts that are updated with new events.

For example, **cook_for** is set to 1 hour (3,600 seconds), **cook_for_extension** is set to 30 minutes (1,800 seconds), and **max_cook_for** is set to 2 hours (7,200 seconds). If Cookbook receives a new related alert 40 minutes after the Recipe started clustering alerts, the Recipe extends the total clustering time by 30 minutes from that time to 1 hour and 10 minutes, then:

- If Cookbook receives another alert 1 hour and 5 minutes after the Recipe started clustering, because Cookbook received it within the extended time of 1 hour and 10 minutes, Cookbook further extends the total clustering time to 1 hour and 35 minutes. Cookbook continually extends the total clustering time as it receives more related alerts, provided that they are received within the extended time. Cookbook can extend the total clustering time until the **max_cook_for** time is reached. If Cookbook receives further related alerts after the **max_cook_for** time of 2 hours has elapsed, the Recipe resets and adds them to a new cluster.
- If Cookbook does not receive any further alerts, it stops clustering alerts after the extended time of 1 hour and 10 minutes elapses. If Cookbook then receives another alert after this time has elapsed, the Recipe starts a new cluster.

You can set a different **cook_for_extension** time for a Recipe, and this overrides the Cookbook value. Recipes without **cook_for_extension** values inherit the value from the Cookbook.

Type: Integer

Required: No

Default: **"1000"**

max_cook_for

Maximum time period, in seconds, that Cookbook can extend clustering alerts for before the Recipe resets and starts a new cluster. It is used in conjunction with the **cook_for_extension** to help to ensure that Cookbook continues to cluster alerts together that are related to the same failure. This value is ignored unless **cook_for_extension** is specified. If **max_cook_for** is not specified, it defaults to three times the **cook_for** period.

Type: Integer

Required: No

Default: 3 x **cook_for** value

Recipes

Recipes determine how Cookbook detects relationships between alerts and considers them for clustering into Situations. You can configure Recipes with different event filters, triggers and similarity comparisons using these parameters:

chef

The recipe type: CValueRecipeV2, CValueRecipe or CBotRecipe. The Value Recipes cluster according to the recipe definitions whereas Bot Recipes follow custom clustering logic defined by a Moobot. See Configure a Cookbook Recipe for more details.

Type: String

Required: Yes

One of: **`CValueRecipeV2`**, **`CValueRecipe`**, **`CBotRecipe`**

Default: **`"CValueRecipeV2"`**

name

Name of the Recipe. Use a unique or descriptive name.

Type: String

Required: Yes

Default: **`"SplitBySourceAndDescription"`**

description

Description of the Recipe.

Type: String

Required: No

Default: **`"Value Recipe outage"`**

recipe_alert_threshold

Maximum number of alerts to cluster before Cookbook creates a Situation. If left as '0',a single alert can generate a new Situation.

Type: Integer

Required: Yes

Default: **`0`**

exclusion

Filter that determines the alerts to exclude from Situation creation. By default, Cookbook excludes all alerts with a severity less than critical. For details on creating a filter, see Filter Search Data.

Type: String

Required: No

Default: **`"severity < 5"`**

trigger

Filter that determines the alerts that Cookbook considers for Situation creation. Cookbook ignores alerts that match the **`exclusion`** filter.

Type: String

Required: No

Default: **`"null"`**

seed_alert

Filter that determines whether to create a Situation from a seed alert if it meets both **`trigger`** and **`seed_alert`** filter criteria. Cookbook considers subsequent alerts for clustering if they meet the **`trigger`** filter criteria. Alerts that arrived prior to the seed alert that met the **`trigger`** filter criteria do not form Situations.

The **`seed_alert`** filter is a mechanism to ensure that only specific events create Situations. For example, if you create a **`seed_alert`** filter if the description matches 'Switch failure', alerts are eligible for clustering only after a seed alert with the matching description arrives to create a Situation.

Type: String

Required: No

Default: **`"null"`**

Example: **`'Description' MATCHES "Switch failure"`**

rate

Filter that determines the minimum event rate per minute required for Cookbook to create a Situation. Cookbook only calculates the rate after the cluster meets the threshold defined by **`min_sample_size`** or **`max_sample_size`**.

Type: Integer (Number of events per minute).

Required: No

Default: **`"0"`**

min_sample_size

Minimum number of events contained in a cluster before Cisco Crosswork Situation Manager calculates the rate.

Type: Integer

Required: No

Default: **`"5"`**

max_sample_size

Maximum number of events contained in a cluster before Cisco Crosswork Situation Manager calculates the rate.

Type: Integer

Required: No

Default: **`"10"`**

cluster_match_type

Defines how Cookbook matches alerts to clusters. The **`first_match`** default option adds alerts to the first cluster above the similarity threshold value. The alternative is **`closest_match`** to add alerts to the cluster with the highest similarity greater than the similarity threshold value. The latter option might be less efficient because it needs to compare alerts against each cluster in a Recipe.

Type: String

Required: No

Default: **`"first_match"`**

cook_for

Minimum time period, in seconds, that Cookbook clusters alerts for before the Recipe resets and determines when to start a new cluster. Different **cook_for** times per Recipe are useful for monitoring systems with different fail rates, to ensure the Recipe clusters all the relevant events relating to a failure. For example:

- A Recipe monitoring for network link failures, which have a fast fail rate and many events in a short time, should have a short **cook_for** time.
- A Recipe monitoring for disc or CPU issues, which have a slower fail rate as the issue builds, should have a longer **cook_for** time.

If you set a different **cook_for** time for a Recipe, this overrides the Cookbook value. Recipes without **cook_for** values inherit the value from the Cookbook.

Type: Integer

Required: No

Default: **"5000"**

cook_for_extension

Time period, in seconds, that Cookbook can extend clustering alerts for before the Recipe resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Recipe. As Cookbook receives related alerts, it continues to extend the total clustering time until the **max_cook_for** period is reached. Used in conjunction with the **max_cook_for** value, the **cook_for_extension** helps to ensure that Cookbook continues to cluster alerts together that are related to the same failure. The **cook_for_extension** only applies to new related alerts; it does not apply to existing alerts that are updated with new events.

For example, **cook_for** is set to 1 hour (3,600 seconds), **cook_for_extension** is set to 30 minutes (1,800 seconds), and **max_cook_for** is set to 2 hours (7,200 seconds). If Cookbook receives a new related alert 40 minutes after the Recipe started clustering alerts, the Recipe extends the total clustering time by 30 minutes from that time to 1 hour and 10 minutes, then:

- If Cookbook receives another alert 1 hour and 5 minutes after the Recipe started clustering, because Cookbook received it within the extended time of 1 hour and 10 minutes, Cookbook further extends the total clustering time to 1 hour and 35 minutes. Cookbook continually extends the total clustering time as it receives more related alerts, provided that they are received within the extended time. Cookbook can extend the total clustering time until the **max_cook_for** time is reached. If Cookbook receives further related alerts after the **max_cook_for** time of 2 hours has elapsed, the Recipe resets and adds them to a new cluster.
- If Cookbook does not receive any further alerts, it stops clustering alerts after the extended time of 1 hour and 10 minutes elapses. If Cookbook then receives another alert after this time has elapsed, the Recipe starts a new cluster.

If you set a different **cook_for_extension** time for a Recipe, this overrides the Cookbook value. Recipes without **cook_for_extension** values inherit the value from the Cookbook.

Type: Integer

Required: No

Default: **"1000"**

max_cook_for

Maximum time period, in seconds, that Cookbook clusters alerts for before the Recipe resets and starts a new cluster. It is used in conjunction with the **cook_for_extension** to help to ensure that Cookbook continues to cluster alerts

together that are related to the same failure. This value is ignored unless **cook_for_extension** is specified. If **max_cook_for** is not specified, it defaults to three times the **cook_for** period.

If you set a different **max_cook_for** time for a Recipe, this overrides the Cookbook value. Recipes without **max_cook_for** values inherit the value from the Cookbook.

Type: Integer

Required: No

Default: 3 x **cook_for** value

Matcher

hop_limit

Maximum number of hops between the alert source nodes in order for the alerts to qualify for clustering. Cisco Crosswork Situation Manager measures hop limit from the first alert that formed the Situation and always follows the shortest possible route in the network. You can only use hop limit if you have imported your network topology into the system. See Import a Topology for details.Import a Topology

A hop is the jump between two directly connected nodes in a network. For more information on hops, see Vertex Entropy.

Type: Integer

Required: No

Default: **"2"**

components

Values that alerts must match for Cookbook to include them in a Situation. You can provide multiple values such as source, description, service or using custom_info fields.

The Value Recipe V2 uses the **shingle_size** component to determine the similarity between different strings. See Recipe Types for more details.

You can enable or disable case sensitivity with CValueRecipe V1. You can also configure Cookbook to match each value in the list individually. See CValue Recipefor details.

Type: String

Required: No

Default:

```
{ name: "source_id", similarity: 0.75, shingle_size: 4 },
   { name: "description", similarity: 0.75, shingle_size: -1 }
```

Classic

The Sigaliser Moolet, also known as Sigaliser Classic, is where in event processing, alert streams from the Alert Builder or the Alert Rules Engine are converted into Situations.

The Sigaliser is self-contained and has no Moobot. It takes every occurrence of an event in an alert stream and uses matrix factorisation algorithms to identify clusters of alerts that are temporally correlated identifying underlying

service outages or Situations. The Sigaliser then updates its own internal knowledge of the stores of the Situations and the Cisco Crosswork Situation Manager database before putting updates out on the Message Bus.

Basic Concepts

You can configure and tune Sigaliser Classic by editing the parameters in the **$MOOGSOFT_HOME/config/moolets/sigaliser.conf** configuration file. Generally, the types of Situations created for a given set of alerts are dependent on the rate of occurrence of alerts. You correspond by adjusting the resolution of the window of the Sigaliser parameters to try and match the activity.

The algorithms work by spotting signature scatter pattern of alerts with in a time period. Firstly, how many optimal clusters there are, which should correspond to the number of current, active, service threatening outages in the given window that the Sigaliser operates on. Secondly, it then optimally factorises it down into individual groups, which Cisco Crosswork Situation Manager calls Situations. Once you have a Situation, a Situation Room is created in the Cisco Crosswork Situation Manager database, and you are notified through the Situation View in the user interface.

The algorithm is run in semi real-time and is triggered by either:

- A fixed polled time period.
- A single time slice being filled up, the width of which is set by the resolution parameter in the configuration. For example, the first alert that arrives after the current slice has been filled will trigger the Sigaliser to run its algorithms.

Sigaliser Configuration

You can define the Sigaliser behavior in the **Sigaliser** section of the Moogfarmd configuration file. In general, the following parameters can be configured to either produce more Situations with fewer alerts, or, fewer Situations with more alerts. The consequence of having more Situations with fewer alerts is that the same underlying outage could be split across multiple Situations. Fewer Situations with more alerts results in the same Situation containing alerts from multiple service outages. The process of tuning the Sigaliser parameters leads to an optimal configuration, where, Situations sharply reflect the state of the managed systems. Cisco refers to Situations being “sharp” and well “resolved” when the parameters give you the best fit of Situations to service outages.

Sigaliser contains a number of properties. The **name**, **classname**, and **run_on_startup** properties are shared with other Moolets.

```
{
        name                            : "Sigaliser",
        classname                       : "CSigaliser",
        run_on_startup          : false,
        process_output_of       : "AlertBuilder"
}
```

name

The **name** is hardcoded and should never be changed from Sigaliser.

classname

The **classname**, **CSigaliser**, is hardcoded and should never be changed.

run_on_startup

By default, **run_on_startup** is set to false, so that when Moogfarmd starts, it does not automatically create an instance of the Sigaliser. In this case you can start it using **farmd_ctrl**.

Undertaking the sigalising

These properties in the Moolet direct which output should be processed:

process_output_of

Instructs the Moolet to process the output of the Alert Builder or Alert Rules Engine. Usually the Sigaliser connects directly to the Alert Builder, and the Alert Rules Engine is only used if automations are desired prior to Situation resolution. The Sigaliser can have only one input.

Algorithmics

The Sigaliser runs the matrix factorization algorithms, the properties for which are as follows:

```
# Algorithm
   time_compression                : true,
   alert_threshold                 : 2,
   membership_limit                : 3,
   sig_similarity_limit            : 0.7,
   sig_alert_horizon               : 0.5,
   scale_by_severity               : false,
   entropy_threshold               : 0.0,
```

time_compression

If set to **true**, the algorithm will ignore any empty time buckets in the Sigaliser calculation. If set to false, it will include the empty time buckets. We recommend that you set **time_compression** to true for low data rates and false for Bu1_Bullet1 data rates.

You only require **time_compression** in scenarios where the data rate is very low when compared to the values of **window** and **resolution**. In certain low data-rate scenarios it is possible for a **window** or **resolution** to contain no alerts. For example, if the data rate is two alerts per hour and the **window** is 15 minutes, on average, some of the time buckets in any Situation calculation will be empty. When **time_compression** is **true** empty time-buckets are removed from the calculation, but the total number of buckets used in the calculation remains the same.

alert_threshold

Defines the minimum number of alerts that a Situation can contain. So, increasing the **alert_threshold** will reduce the total number of Situations. We recommend an **alert_threshold** of 2.

alert_threshold can be used in conjunction with small values of **membership_limit** to produce a smaller number of Situations, each of which has more alerts.

membership_limit

The Situation creation process contains multiple steps, including a resolution and merging step. During the merging phase, the raw Situations from the factorization calculation are compared and merged with the currently active Situations. This detects when a detected Situation is either novel or an evolution in time of an existing Situation.

The **membership_limit** property restricts the number of Situations in which an alert can appear. As Situations become merged with each other over time, it is possible for an alert to appear in more Situations than are defined by

membership_limit. Changing the value of **membership_limit** does not have a large impact on the total number of Situations but does change the distribution of the number of alerts in each Situation.

Decreasing the **membership_limit** results in fewer Situations with more alerts and more Situations containing a small numbers of alerts. Whereas, increasing **membership_limit** results in, more Situations with a greater number of alerts and fewer Situations containing a small numbers of alerts. Therefore, the optimal value seems to be between one and five, with a recommended **membership_limit** of three.

sig_similarity_limit (Jaccard Similarity Coefficient)

A measure of the similarity between two Situations before they are merged together. The value is the Jaccard Similarity Coefficient (JSC) defined as the ratio of shared alerts between two Situations to total unique alerts in both Situations.

For example, if Situation1 & Situation 2 share two common alerts, each Situation has one unique alert:

JSC = 2 (common alerts) / [1 (unique to Situation 1) + 2 (common to both) + 1 (unique to Situation 2)] = 2/(1+2+1) = 2/4 = 0.5.

Reducing the similarity index will reduce the total number of Situations. Smaller values increase the likelihood of Situations being merged together, as they have to share fewer alerts in common to be viewed as the same Situation. Conceptually, JSC values less than 0.5 are hard to justify as grounds for merging, so should be used with care. We recommend a **sig_similarity_limit** of 0.7.

sig_alert_horizon

When the Sigaliser algorithm initially identifies a Situation, it will contain alerts that are more representative of the Situation than others. This parameter, which takes the value between 0.0 and 1.0, allows you to provide a cut off for membership based upon the highest significant alert in the cluster. If you set this value to be 0.5, for example, only alerts that have a “significance” for the Situation that is more than half of the most significant alert in the Situation will be included. 0.5 is the default value.

entropy_threshold

The value of this parameter is the minimum entropy that an alert must possess to be included in the Sigaliser calculation. Any alert that arrives at the Sigaliser with entropy below this value will never be included in a Situation. It has a value between 0.0 and 1.0 and has a default of 0.0 which means every alert will be processed.

scale_by_severity

scaleBySev allows you to bias Cisco Crosswork Situation Manager so that high severity alerts are treated as having higher entropy. If you had the same alert arrive with a critical severity, versus a minor severity, you would give the critical severity the higher entropy than the minor severity. This scaling is done as the severity constant number divided by the maximum severity (5). So in the case of critical, you get all of the entropy and in the case of minor, you get three fifths of the entropy. In the case of clear you would get an entropy value of 0.0.

Triggers and Time Buckets

The algorithm is run incrementally as events are ingested, as such Situations are produced and updated in real-time. There are two ways to trigger the algorithm: using a time interval or using the rate of the event stream.

```
# Triggers
   sig_on_bucket   : true,
   sig_interval    : 100,
   max_backlog     : 1000000,
   # Time Buckets
   resolution              : 120,
   window                  : 90
```

The optimal trigger for production should be **sig_on_bucket=true**, provided this ensures satisfactory Situation accuracy and that Situations are being regularly updated. **sig_on_bucket** can also simulate real-time behavior using historical data.

When Situations are not being updated regularly enough, configure **sig_on_bucket** = **false** and set **sig_interval** to a value no more than half of the real-time size of the window.

In a production environment, set **max_backlog** to a high value to avoid triggering the Sigaliser between timed executions. This parameter will cause the algorithms to run if the number of events that arrive before either a scheduled execution, or a bucket being filled is above this value. It should be used with care and only when you have an environment where the event rate is highly variable.

sig_on_bucket

If set to **true**, the Sigaliser will run whenever a new time bucket occurs. Depending upon the data rate, this has the effect of executing the Sigaliser after every defined number of “resolution” seconds.

sig_on_bucket = true deactivates both the **sig_interval** and **max_backlog** triggers.

sig_interval

Executes the Sigaliser algorithm every defined number of seconds, in the example above, every 100 seconds.

sig_interval and **max_backlog** do not override each other; consequently, it is possible for the Sigaliser to be executed more frequently through the **sig_interval** value.

max_backlog

Executes the Sigaliser if the number of defined Alerts are received since last execution, in the example above, the Sigaliser is executed after 1,000,000 alerts are received.

resolution

The duration, in seconds, for each bucket of time that the event stream is divided into. A high value for the resolution will result in Situations that are less “sharp” in time, as the wider the bucket the more likely that alerts from disconnected outages will occur in the same bucket, and potentially in the same Situation.

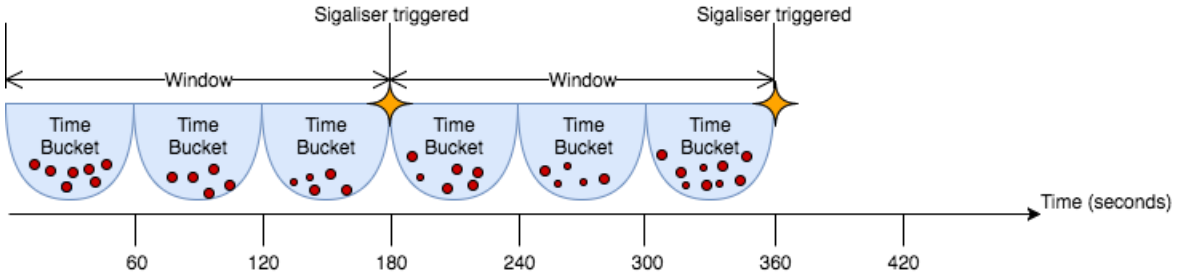
window

The number of time-buckets to include in the calculation. The width of the window should be chosen to match the average time period over which outages typically evolve. The total amount of time considered in any Sigaliser calculation is window multiplied by the **resolution**.

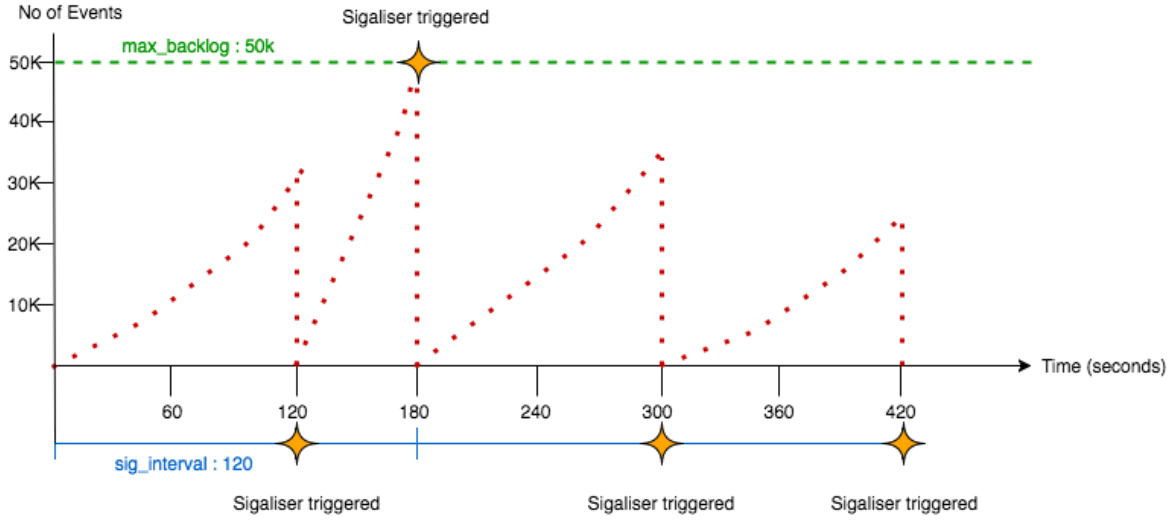
In general, for a high data rate you would use a smaller **resolution** and **window** than for a low data rate. For a fixed data rate, a smaller **resolution** will generally result in more Situations.

Diagrams

The diagram below illustrates how a Sigaliser can be triggered every 180 seconds if 'sig_on_bucket' is set to 'true', the time bucket resolution is set to '60' and the window is '3':



The diagram below illustrates how a Sigaliser can be triggered if 'sig_interval' is set to 120 seconds and if 'max_backlog' is set to 50,000 events:



Feedback

Warning: Feedback is a prototype feature and is not recommended or supported for use in production environments.

Feedback is a clustering algorithm that creates Situations by learns from feedback that users add in the Cisco Crosswork Situation Manager UI.

You can configure Feedback by training it to identify the type of alerts you want it to cluster into Situations. When Feedback creates a Situation, it creates a neural network or "brain" that remembers the contents of the Situation. You can train each brain by identifying which alerts you want Feedback to include and exclude from Situations.

Feedback learns and unlearns by example, remembering what users identify as Situations and what it should not include in future Situations. For example, if Feedback creates a seed Situation with alerts for a host down error and a database restart error with the same IP address, it also creates a brain for that Situation. If Feedback clusters three new alerts based on this brain and you want to delete one of these alerts from the Situation, this retrains the brain.

The main risk with Feedback is if you provide it with erroneous feedback, the Situations it produces will also be erroneous.

To configure Feedback, see Configure Feedback.

Configure Feedback

Warning: Feedback is a prototype feature and is not recommended or supported for use in production environments.

Feedback is a supervised machine learning algorithm that creates Situations based on user feedback.

You can enable and configure Feedback in the **`$MOOGSOFT_HOME/config/moolets/feedback.conf`** configuration file. After you enable Feedback, you can select and edit the Feedback brain via the System Settings in the Cisco Crosswork Situation Manager UI.

Refer to Feedback Reference to see all available properties.

Before You Begin

Before you set up Feedback, ensure you have met the following requirements:

- Your LAMs or integrations are running and Cisco Crosswork Situation Manager is receiving events.
- You have configured the Moolet that is the source of the alerts for Feedback. You select the source using the **`process_output_of`** property.

Configure Feedback

Edit the configuration file at **`$MOOGSOFT_HOME/config/moolets/feedback.conf.`**

See Feedback Reference for a full description of all properties. Some properties in the file are commented out by default.

1. Provide a name for algorithm:
   - name:Name of the Feedback Moolet.
2. Configure Feedback's behavior for when it starts and stops running:
   - run_on_startup: Determines whether Feedback runs when Cisco Crosswork Situation Manager starts.
   - process_output_of: Defines the Moolet sources of the alerts that Feedback processes.
3. Configure the Feedback algorithm and how it clusters alerts:
   - membership_limit:Maximum number of Situations that an alert can be a member of.
   - scale_by_severity: Feedback treats alerts with a high severity like alerts with a high entropy value.
   - entropy_threshold: Minimum entropy value an alert must have in order for Feedback to include it in a Situation.
   - single_matching: Match alerts to the most suitable neural network.
4. Configure the neural networks:
   - inputs:Alert attributes that you are interested in matching and want Feedback to consider when learning a Situation.
   - learn_queues: Collection of actions you want to trigger learning.
   - unlearn_queues: Collection of actions that can trigger the removal of a neural network.
   - rating_threshold:Measures when to trigger learning or unlearning.

- match_strategy: Determines how Feedback matches alerts.
- precision: Determines how precisely you want to train the brain.
- tolerance: Determines the degree of error an alert can have to create a new Situation.
- window: Determines the length of time in seconds that Feedback analyzes alerts and develops a Situation each time it runs.

Restart the Moogfarmd service to activate any changes you make to the configuration file. See Control Moogsoft AIOps Processes for further details.Control Moogsoft AIOps Processes

Example

The following example demonstrates a simple Feedback Sigaliser:

```
{
            # Moolet
            name                : "Feedback",
            classname           : "com.moogsoft.farmd.moolet.feedback.CFeedback",
            run_on_startup      : false,
            process_output_of   : "MaintenanceWindowManager",
            membership_limit    : 1,
            scale_by_severity   : false,
            entropy_threshold   : 0.0,
            single_matching     : false,
            inputs              : [ "source","description" ],
            learn_queues        : [ "manual_create","rated" ],
            unlearn_queues      : [ "rated" ],
            rating_threshold    : 3,
            precision           : 92.0,
            tolerance           : 0.01,
            exact_match         : false,
            window              : 60
    }
```

Feedback Reference

Warning: Feedback is a prototype feature and is not recommended or supported for use in production environments.

This is a reference for the Feedback clustering algorithm. The Feedback configuration properties are found in **$MOOGSOFT_HOME/config/moolets/feedback.conf.**

Cisco recommends you do not change any properties that are not in this reference guide.

Moolet

name

Name of the Feedback Sigaliser algorithm. Do not change.

Type: StringRequired: Yes

Default: **"Feedback"**

class

Moolet class name. Do not change.

Type:StringRequired: Yes

Default:**"com.moogsoft.farmd.moolet.feedback.CFeedback"**

run_on_startup

Determines whether Feedback runs when Cisco Crosswork Situation Manager starts. If you enable this property, Feedback captures all alerts from the moment the system starts, without you having to configure or start it manually.

Type: Boolean

Required: No

Default: **false**

process_output_of

Defines the Moolet source of the alerts for Feedback.

Type: List

Required: Yes

One of: `AlertBuilder, AlertRulesEngine, MaintenanceWindowManager, EmptyMoolet` Default: "**MaintenanceWindowManager"**

Algorithm

membership_limit

Maximum number of Situations an alert can be part of. This does not impact alerts in merged Situations. Smaller limits result in fewer Situations with many alerts and many Situations with fewer associated alerts. Larger limits result in many Situations with few alerts and a few Situations with many alerts. The optimal value is between 1 and 5.

Type: Integer

Required: Yes

Default: 1

scale_by_severity

Feedback treats alerts with a high severity like alerts with a high entropy value. Cisco Crosswork Situation Manager divides the severity number by the maximum severity (5) to calculate the scale. For example, for an alert with minor severity, the entropy is 3/5.

Type: Boolean

Required: No

Default: False

entropy_threshold

Minimum entropy value that an alert must have for Feedback to consider it for clustering into a Situation. Feedback does not include any alerts with an entropy value below the threshold in Situations. Set to a value between 0.0 and 1.0. The default of 0.0 means Feedback processes all alerts.

Type: Decimal

Required: No

Default: 0.0

single_matching

Enable **`single_matching`** for Feedback to match alerts to the most suitable neural network.

Type: Boolean

Required: No

Default: **`false`**

Neural Network (Brain)

inputs

Alert attributes that you are interested in matching and want Feedback to consider when learning from Situations.

Type: Array

Required: Yes

Default: **`[ "source","description","severity","manager" ],`**

learn_queues

Collection of actions that you want to trigger learning. See User Actions for available actions. For example, you might want Feedback to learn when a user creates a Situation or gives a Situation a high rating.

Type: Array

Required: Yes

Default: **`[ "manual_create","rated","merge_create","split_create","annotated","diagnosed","refined"],`**

unlearn_queues

Collection of actions that can trigger the removal of a neural network. For example, you might want Feedback to unlearn when a user gives a Situation a low rating.

Type: Array

Required: Yes

Default: **`[ "rated","split" ],`**

rating_threshold

Measures when to trigger learning or unlearning determined by the star rating out of five that users give the Situations in the Cisco Crosswork Situation Manager UI. By default, a rating of three or higher triggers learning. A rating of lower than three triggers unlearning.

Type: Array

Required: Yes

Default: **3**

precision

Determines how precisely you want to train the brain. Increase this value if Feedback is not producing accurate results. Cisco does not recommend reducing this below 92%.

Type: Integer (%)

Required: Yes

Default: **92.0**

tolerance

Determines the percentage degree of error an alert can have to create a new Situation. For example, if the neural network has a trained value of 0.95 and the new alert has a value of 0.8, the tolerance needs to be 0.15 or lower for Feedback to create a new Situation.

Type: Integer (%)

Required: Yes

Default: **0.01**

window

Determines the length of time in seconds that Feedback analyzes alerts and a Situation develops each time it runs. Defaults to 60 seconds.

Type: Integer

Required: Yes

Default: **60**

User Actions

Feedback can learn or unlearn from the following user actions:

- **manual_create**: User has manually brought alerts together to create a Situation.
- **merge_create**: User has merged two or more Situations together. Feedback learns the newly created Situation.
- **split_create**: User has created some new Situations by splitting an existing Situation. Feedback learns from the newly created Situations.
- **annotated**: User has started a discussion thread or added a comment on the Situation or has set a Situation's description.
- **diagnosed**: Tool has been run on the Situation.
- **closed**: Situation has been closed or resolved.
- **refined**: Alerts have been added to or removed from a Situation.

Tempus

Tempus is the time-based algorithm in Cisco Crosswork Situation Manager which clusters alerts into Situations based on the similarity of their timestamps.

The underlying premise of Tempus is that when things go wrong, they go wrong together. For example, if a core element of your network infrastructure such as a switch fails and disconnects then it affects a lot of other interconnected elements and send events at a similar time.

Tempus uses the Jaccard index to calculate the similarity of different alerts. It also uses community detection methods to identify which alerts with similar arrival patterns it should cluster into Situations.

As Tempus is time-based, you should not be use it to detect events relating to the slow or gradual degradation of a service from disks filling up or CPU usage.

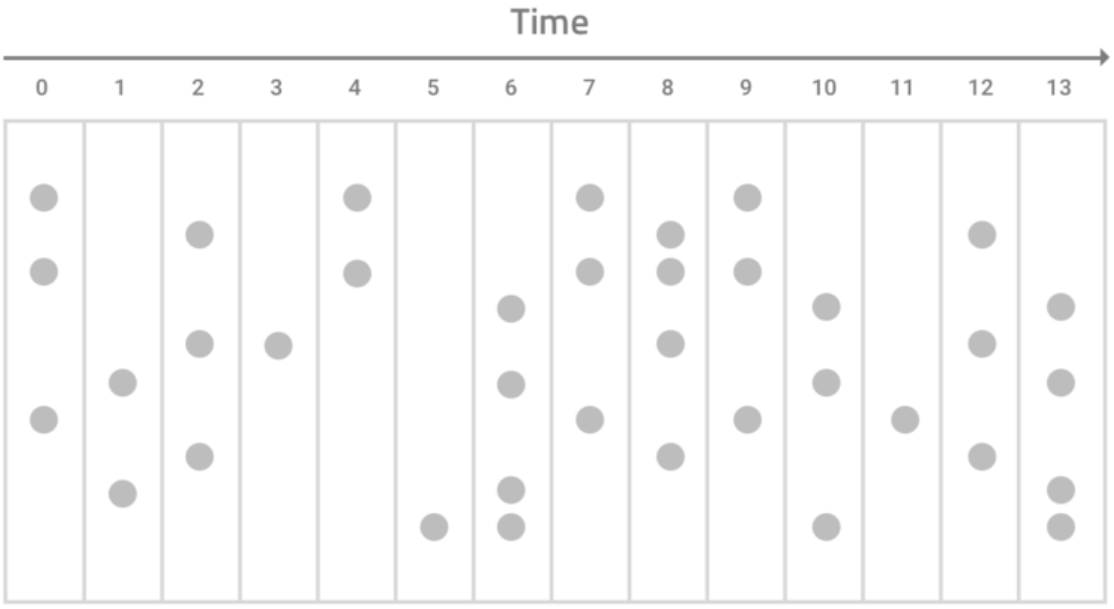
Note: One advantage of Tempus is it only uses event timestamps for clustering so no alert enrichment is required.

Time-based Clustering

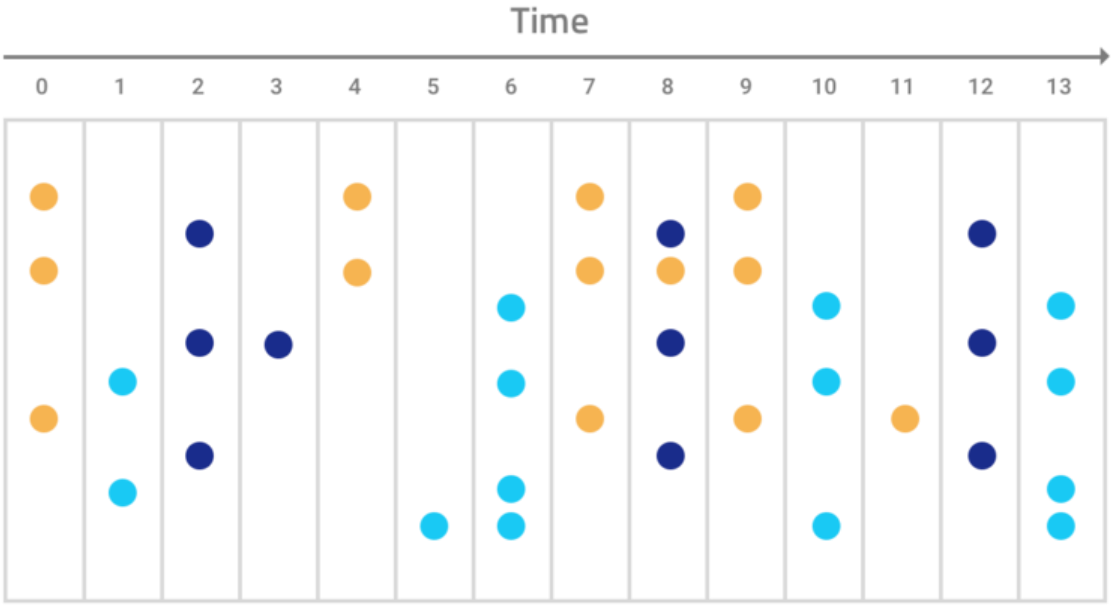
Cisco Crosswork Situation Manager applies Tempus incrementally to alerts as it ingests them so that it can create Situations in real-time.

The diagrams below show how Tempus sorts and the groups alerts with similar timestamps into Situations:

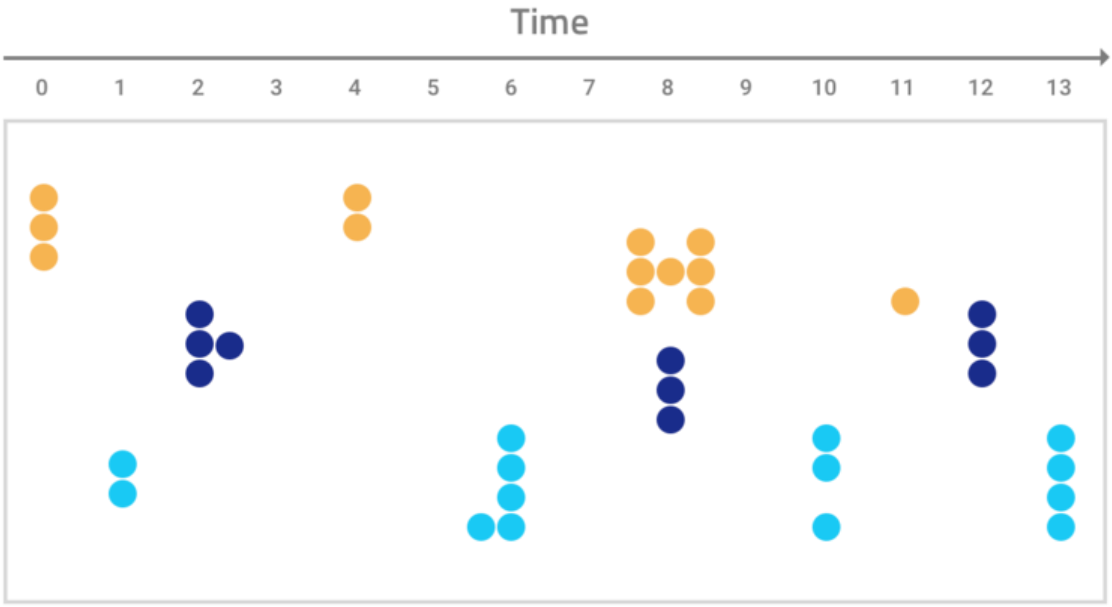
Raw alerts from either the Alert Builder or Alert Rules Engine arrive over a period of time. These are shown as gray dots in the diagram below:



Tempus identifies and sorts which alerts have similar arrival patterns:



Alerts with similar arrival patterns are clustered into Situations:



Configure Tempus

You can configure and tune Tempus in **`$MOOGSOFT_HOME/config/moolets/tempus.conf`**. The Moolet parameters configure general information about each Sigaliser. The Output parameters control where the output processed by Tempus originates from. The Trigger and Sigalising parameters control the Sigaliser execution and duration.

Moolet Parameters

The parameters that relate to the Tempus Moolet are as follows:

run_on_startup

Determines whether Tempus runs when Cisco Crosswork Situation Manager starts. If enabled, Tempus captures all alerts from the moment the system starts, without you having to configure or start it manually.

Type: Boolean

Default: **`false`**

metric_path_moolet

Determines whether Tempus is included in the metric for Self Monitoring or not.Self Monitoring

Type: Boolean

Default: **`false`**

description

Describes the Situation produced by the Tempus clustering algorithm.

Type: String

Default: **`A Tempus (a.k.a. Sigaliser V2) Situation`**

The default Tempus parameters are as follows:

```
name              : "Tempus",
   classname         : "com.moogsoft.farmd.moolet.tempus.CTempus",
   run_on_startup    : false,
   metric_path_moolet   : true,
   #process_output_of : "AlertRulesEngine",
   process_output_of : "AlertBuilder"
   description       : "A Tempus (a.k.a. Sigaliser V2) Situation",
```

Note: name and classname are hard coded and should not be changed.

Output Parameters

These parameters control the output processed by the Tempus clustering algorithm:

process_output_of

Defines the Moolet source of the alerts that Tempus processes. By default, the Sigaliser connects directly to the Alert Builder and Alert Rules Engine is only being used if automations are desired prior to Situation resolution.

Type: List

One of: **`AlertBuilder`**, **`AlertRulesEngine`**, **`MaintenanceWindowManager`**, **`EmptyMoolet`**

Default: **`AlertBuilder`**

entropy_threshold

Sets the minimum entropy value for an alert to be clustered into a Situation. Tempus does not include any alerts with an entropy value below the threshold in Situations. Set to a value between 0.0 and 1.0. The default of 0.0 means all alerts are processed.

Type: Integer

Default: **0.0**

The default output parameters are as follows:

```
# process_output_of  : "AlertRulesEngine",
   process_output_of    : "AlertBuilder"
   description          : "A Tempus (a.k.a. Sigaliser V2) Situation",

   # Algorithm
   entropy_threshold    : 0.0,
```

Trigger and Sigalising Window Parameters

The execution and duration of Tempus is controlled by the trigger, window and bucket parameters:

- The sig_interval trigger determines when Tempus starts to run
- The window is the total span of time in seconds in which alerts will be analyzed each time Tempus runs
- Time buckets are small five-second subdivisions of the window in which the Alerts are captured.

sig_interval

Executes the Tempus algorithm after a defined number of seconds. In the example above, the Sigaliser will run every 120 seconds (two minutes).

Type: Integer

Default: 120

window_size

Determines the length of time of the window in which alerts are analyzed and a Situation develops each time the Sigaliser is run. By default the Sigalising window is 1200 seconds (20 minutes).

Type: Integer

Default: **1200**

bucket_size

Determines the time span of each bucket in which alerts are captured in seconds. By default each bucket is five seconds long so there will be 240 buckets per window.

Type: Integer

Default: **5**

Warning: Cisco does not recommend you change the bucket size. If you do want to change the bucket_size then change with caution because Tempus is designed to use small bucket sizes.

arrival_spread

Sets the acceptable latency or arrival window for each alert in seconds. This can be used to minimise or reduce the impact of multiple alerts arriving over a small amount of time and landing in separate buckets.

Type: Integer

Default: **15**

min_arrival_similarity

Determines how similar alerts must be to be consider for clustering. This is useful way to determine what proportion of the events two alerts need to share to have a similar pattern of arrival. By default this is 0.6667 which means Tempus will disregard any alerts with less than two-thirds similarity.

Type: Integer

Default: **0.6667**

The default trigger and sigalising window parameters are as follows:

```
      # Triggers
         sig_interval      : 120,    # seconds => sigalise every 2 minutes

         # Sigalising Window
         window_size       : 1200,   # seconds => 20 minutes
         bucket_size       : 5,      # seconds : Take Care if changing - Tempus is
designed to use small bucket sizes
         arrival_spread    : 15,     # seconds : acceptbale latency/arrival window
for each event
```

Partitioning

Partitioning is set to 'null' by default. There are two methods to partition data into Situations. The first is 'partition_by' which splits the clusters according to the parameters specified. The second is 'pre_partition', which splits the incoming event stream before clustering.

partition_by

After clustering has taken place and before you enter merging and resolution, you can split clusters into sub-clusters based on a component of the events. For example, you can use the **manager** parameter to ensure the Situations only contain events from the same manager. In general, and by default, you should comment out the **partition_by** parameter.

Note: Pre-partitioning is recommended as it does not interfere with the results of the clustering algorithms.

pre_partition

An alternative way of partitioning is to use **pre_partition** which allows you to specify a component field (from the list of specified components) around which the event stream will be partitioned before clustering occurs. The Alerts in the resulting Situations will each contain a single value for the component field chosen.

Significance

You can configure Tempus to only create Situations from alerts that meet a certain degree of constant significance based upon Poisson distribution calculations.

significance_test

Calculation that determines how significant a cluster of alerts or potential Situation must be for Tempus to detect it. The default, **Poisson1**, looks at the data of a single alert cluster to calculate how significant it is. The default is more likely to detect all significant alert clusters but with a higher risk of creating insignificant alert clusters. Use this option when your alerts originate from different networks or unrelated topologies. **Poisson2** is a more thorough test that looks at an alert cluster and all alerts outside the cluster with a similar event rate. It is more likely to exclude all insignificant alert clusters but with a high risk of excluding significant alert clusters. Use this option if you expect all of your alerts to come from the same connected network.

Type: String

One of: **Poisson1**, **Poisson2**

Default: **Poisson1**

significance_threshold

Sets the maximum significance score in order for Tempus to create a Situation. The score is proportional to the probability that the alert cluster or potential Situation was coincidence. The lower the score, the more significant the cluster and the least likely it was a coincidence. The **significance_threshold** score ranges from 0-100.

Type: Integer

Default: **1**

Tempus Example

Tempus appears in **$MOOGSOFT_HOME/config/moolets/tempus.conf** as follows:

```
{
            # Moolet
            name                    : "Tempus",
            classname               : "com.moogsoft.farmd.moolet.tempus.CTempus",
            run_on_startup          : false,
        metric_path_moolet  : true,
        #process_output_of  : "AlertRulesEngine",
        process_output_of   : "AlertBuilder"
            description             : "A Tempus (a.k.a. Sigaliser V2) Situation",

            # Algorithm
            entropy_threshold       : 0.0,

            # Triggers
            sig_interval            : 120,    # seconds => sigalise every 2 minutes

            # Sigalising Window
            window_size             : 1200,   # seconds => 20 minutes
            bucket_size             : 5,      # seconds : Take Care if changing - Te
    mpus is designed to use small bucket sizes
            arrival_spread          : 15,     # seconds : acceptbale latency/arrival
    window for each event

            # How similar must alerts be to be considered for clustering?
            min_arrival_similarity : 0.6667,
```

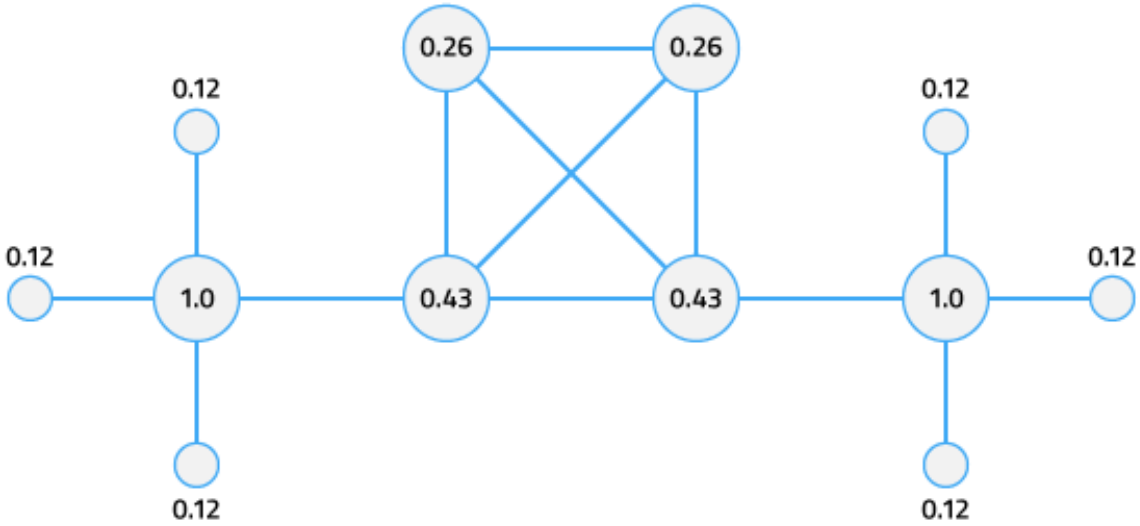
```
            pre_partition           : null,
            partition_by            : null

        significance_test       : "Poisson1"
            significance_threshold  : 1
    }
```

Vertex Entropy

Vertex Entropy is a Cisco Crosswork Situation Manager algorithm that indicates the critical nodes within your network and their tendency to produce important events.

You can use Vertex Entropy if you want to cluster alerts into Situations based on their topological importance. Once the calculation is run against a topological map of the connected nodes in your network, it applies a Vertex Entropy value for each node or "vertex".



This diagram shows the Vertex Entropy values for a network of 12 connected nodes. A Vertex Entropy value of 1.0 indicates a node of highest topological importance.

Note:

Node - a device or base unit that forms part of a larger network. This is known as a 'vertex' in graph theory.

Link - a connection between two directly connected nodes. This is known as an 'edge' in graph theory.

Hop - a jump between two directly connected nodes.

Before You Begin

Before you perform the Vertex Entropy calculation and enable its associated features, ensure you have met the following requirements:

- You have a map of the connected nodes in your network in a comma-separated value (.csv) file.
- Your .csv topology file contains all of the nodes that you expect to send events.

- You have run the topology builder utility to import your network topology. See Import a Topology.

Calculate the Vertex Entropy

To enable the features associated with Vertex Entropy, follow these steps:

1. Import your network topology .csv file into the database using the topology builder found at **$MOOGSOFT_HOME/bin:**

```
./topology_builder -t <file_name>.csv
```

The topology builder utility uses the source data to build a topology. If there is no pre-existing topology, topology builder records host names in the **entity_catalog** table. By default topology builder assigns each node to the 'Network' group and the 'Unix servers' competency. The utility records the topological information in the **moog_reference.one_hop_topo** and **moog_reference.topo_nodes** tables. See Import a Topology for more information.Import a Topology

2. Run the graph analyser from **$MOOGSOFT_HOME/bin** to calculate the Vertex Entropy for the nodes or "vertices" in your network. This is a one-off calculation:

```
./graph_analyser
```

Graph analyser disregards any weight values included in your imported network topology. See Graph Analyser Command Reference.

Once the graph analyser has been run and each node has a Vertex Entropy value, you can start using these values in other areas of Cisco Crosswork Situation Manager.

Add a Vertex Entropy Filter

You can use Vertex Entropy as either a trigger or an exclusion filter in a Cookbook Recipe. These filters include or exclude alerts with similar importance in terms of network connectivity. For example, you can set a trigger so Cookbook considers alerts with a Vertex Entropy value of over 0.5 for Situation creation.

To do this, follows these steps:

1. Create a new Recipe under your Cookbook Moolet in **$MOOGSOFT_HOME/config/moolets/cookbook.conf**
2. Add the following value for the trigger parameter:

```
trigger       :"vertex_entropy > 0.5",
```

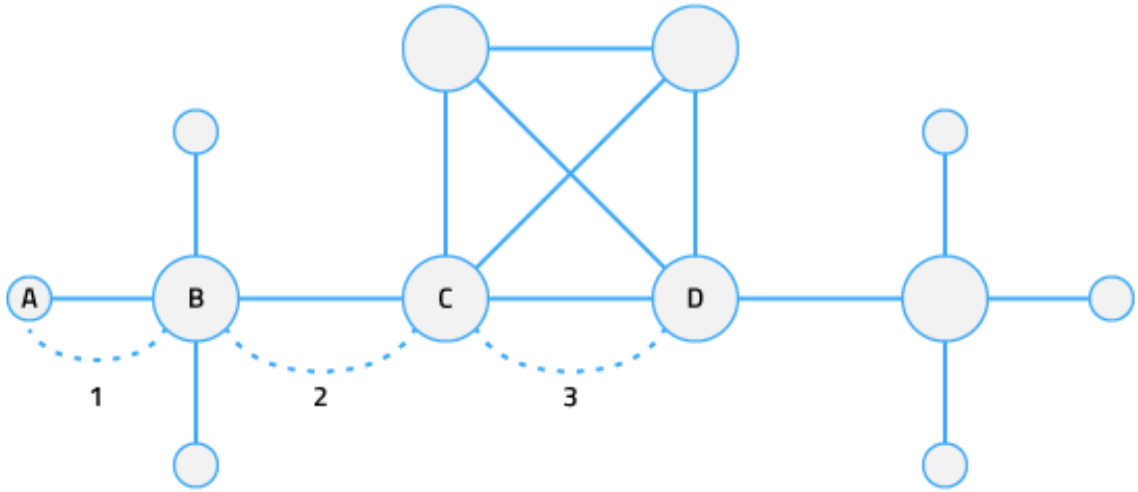
3. Save the changes and restart moogfarmd.

Alternatively, you can create an exclusion filter to exclude alerts with a Vertex Entropy value of less than 0.3 from Situation creation:

```
exclusion :"vertex_entropy < 0.3",
```

Add a Hop Limit

You can add a**hop_limit** filter as part of the matcher configuration in a CValueRecipe so Cookbook clusters alerts from nodes within a certain number of hops from each other. This can be used alongside Vertex Entropy trigger or exclusion filters.



In this diagram, a hop limit of '3' means Cookbook includes alerts from all nodes between node A and node D.

The **hop_limit** filter ensures Cookbook only clusters alerts that originated from nodes that are close together. For more information, see Cookbook configuration.

Add a Seed Alert

You can add a **seed_alert** filter to a Recipe to ensure Cookbook only creates a new Situation if an alert has a specific Vertex Entropy value.

For example, if you only want to create Situations when there is an issue with the most critical nodes in your network, you can set the **seed_alert** filter to only create Situations from alerts with a Vertex Entropy value of 1.0:

```
seed_alert :"vertex_entropy=1.0",
```

If enabled, the initial seed alert must meet both the trigger and **seed_alert** conditions. For more information, see Cookbook configuration.

The **seed_alert** filter is not specific to Vertex Entropy and can be used for other conditions such as severity.

Cookbook Recipe Example

The example Cookbook Recipe below filters for alerts with a high Vertex Entropy:

```
{
        chef: "CValueRecipe",
        name: "VertexEntropy",
        description: "Recipe for alerts with a high Vertex Entropy",
        recipe_alert_threshold: 0,
        exclusion: "vertex_entropy < 0.3",
        trigger: "vertex_entropy > 0.8",
        seed_alert: "vertex_entropy = 1.0",
        rate: 0,
        #Given in events per minute
```

```
        min_sample_size: 5,
        max_sample_size: 10,
        cook_for: 5000,
        cluster_match_type : "first_match",
        matcher: {
                            hop_limit: 2,
                    components: [{
                    name: "source_id",
                    similarity: 1.0
                }
            ]
        }
    }
```

Graph Analyser Command Reference

This is a reference for the **graph_analyser** utility used to calculate Vertex Entropy. The graph_analyser command-line utility accepts the following arguments:

Argument	Input	Description
-h,--help	-	Display the **graph_analyser** utility syntax and option descriptions.
-l, --loglevel	WARN|INFO|DEBUG|TRACE	Log level controlling the amount of information that **graph_analyser** logs. Defaults to INFO.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation*.

To receive new and revised Cisco technical content directly to your desktop, you can subscribe to the What's New in Cisco Product Documentation RSS feed. The RSS feeds are a free service.

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco Trademark

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

Cisco Copyright