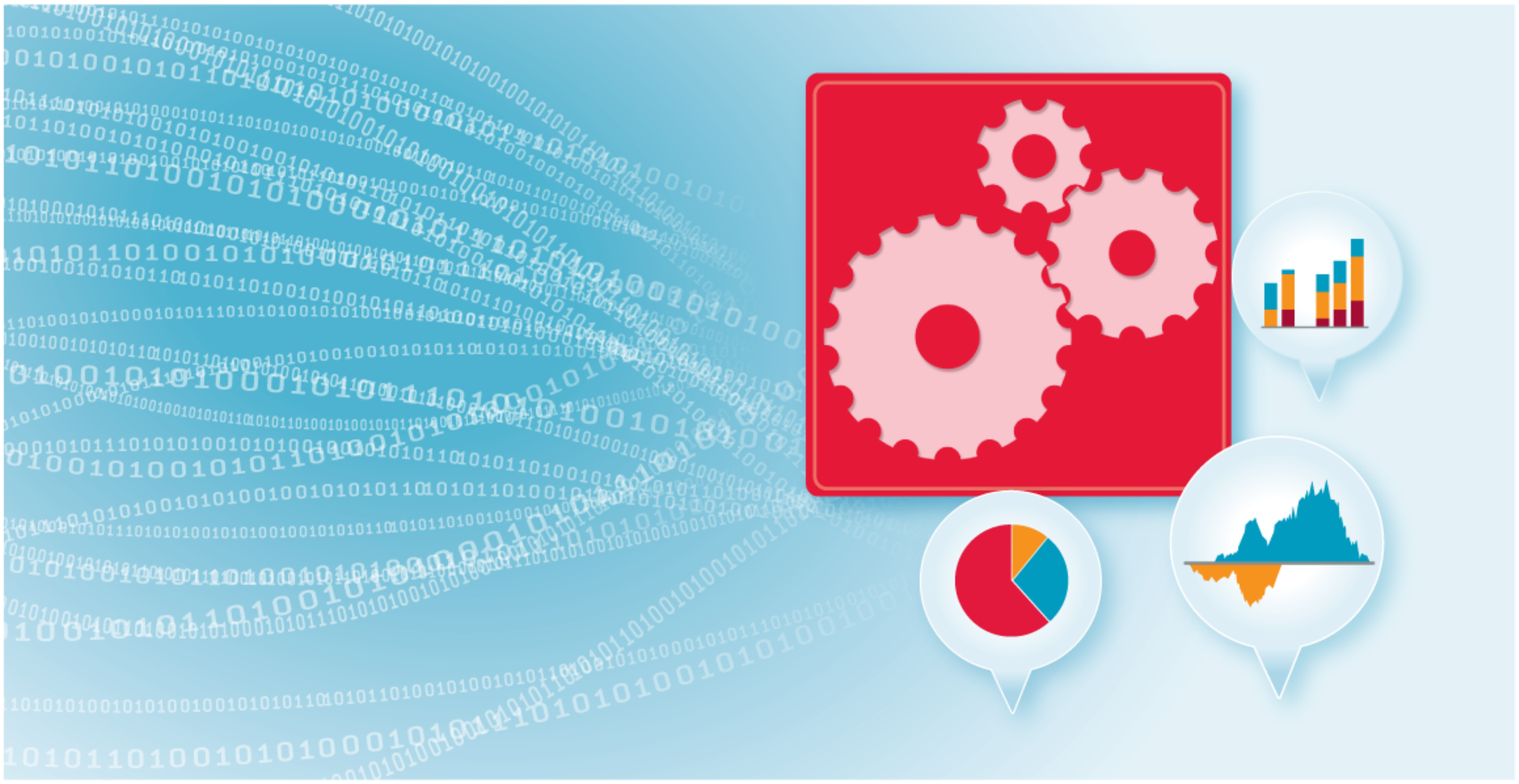

Thread Limit Configuration Guide

Lavastorm Analytics Engine

Legal notice

Copyright

Disclaimer

Table of contents

1. Introduction **4**

2. Defining pools **5**

2.1 LAE's pool assignment method 7

3. Delayed graph execution **8**

4. Automatic restarting of graphs **9**

1. Introduction

To optimize the throughput of the LAE servers, you can impose limits on the number of threads (drones) running on the server at any given time. These limits may be configured to vary for different users or functions.

This section explains the use of pools to govern drone allocation, and describes the optional settings available to handle cases where there aren't sufficient drones available.

Each LAE controller attempts to requisition drones from LAE servers. You can establish pools on each server, each with a limited number of drones. Once the limit of drones available has been reached for the assigned pool and its overflow pools, the server will deny drone creation. This prevents unlimited numbers of processes from running simultaneously on the server, enabling it to operate at optimum throughput.

2. Defining pools

Pools of drones are defined in a pool configuration file. The format of the pool configuration file is similar to other LAE configuration files. There are no reserved names, so the pool name can be any alphanumeric word.

The following table shows the parameters that are available for each pool definition:

Parameter	Value	Note
`pool`	The name of the pool.	Each pool name must be unique.
`size` (optional)	The maximum number of drones available to the pool.	If no value is specified, the pool will be unlimited.
`defaultuser` (optional)	The name of a user who should default here.	Multiple users are specified on multiple lines. A user can only default to one pool.
`defaultgroup` (optional)	The name of a group whose users should default here.	Multiple groups are specified on multiple lines. A group can only default to one pool.
`overflow` (optional)	The name of another pool, to be used if `size` is exceeded.	Multiple lines are allowed. The overflow pool must be defined in the file.
`end`	The name of the pool.	Must match the `pool` parameter.

The following notes explain how these parameters affect drone allocation:

- The `defaultuser` and `defaultgroup` parameters are not security declarations. They are used to determine which pool(s) will be considered by default when the controller does not specify a particular pool.
- Overflow pools can be chained. A pool that *is* an overflow can also *have* an overflow. To prevent circular overflow chaining, a pool can only be used as an overflow if it has already been named in the file.
- Pool selection is made from bottom to top. Define general pools at the top of the file, so that they can be used as overflows.

Example

The following is an example configuration file, with some helpful comments.

Note: Any text that follows a # symbol on a given line is a comment. Comments are purely informational and do not affect the configuration.

```
# An unlimited pool for BRE users
pool:bre
defaultgroup:users
end:bre

# A shared pool for all production graphs
pool:production
size:15
defaultgroup:production
end:production

# An overflow pool for special tiger team members
pool:tiger
size:10
defaultuser:tigermanager
defaultgroup:tigerteam
end:tiger

# A pool for fraud graphs
pool:fraud
size:20
overflow:production
defaultgroup:fraud
end:fraud

# A pool for delinquency graphs
pool:delinquency
size:10
overflow:production
overflow:tiger
defaultgroup:delinquency
end:delinquency
```

Once this file has been created, drone pooling can be turned on by setting the property **ls.brain.server.drone.poolconfig** to the name of the configuration file. This file is re-read every time a new drone is requested allowing the ability to adjust pools and pool assignments dynamically.

Note: Decreasing or removing pools will not cause drones to be terminated. Drones are terminated only when the client (BRE or the LAE Controller) terminates it.

2.1 LAE's pool assignment method

LAE has several steps for reserving drones across a variety of cases:

1. If there is no pool configuration file specified for **ls.brain.server.drone.poolconfig**, drone throttling is turned off. All drone requests will be met.
2. If a pool is specified via **ls.brain.controller.pool**, a drone from that pool is assigned. If the pool is full, it performs a depth-first search of overflow pools and assigns a drone from the first available pool. If this pool and its overflows are full, LAE will not use any other pools and the request is denied.
3. If no pool is specified by the controller but the **user** has a default pool, it assigns a drone from that pool. If the pool is full, it performs a depth-first search of overflow pools to find an available drone. If none are found, it tries step 4.
4. If no pool is specified by the controller but the user is a member of one or more **groups** that have a default pool, it assigns a drone from that pool. If the pool is full, it performs a depth-first search of overflow pools to find an available drone. If that pool and its overflows are full, LAE will not use any other pools and the request is denied.

3. Delayed graph execution

By default, the entire graph will fail if the controller cannot obtain all requested drones. For cases where it is acceptable to run with fewer nodes, LAE allows control of its drone allocation requirements using the following properties:

- **ls.brain.controller.droneAllocationMinimum** (integer, default=100) – The minimum percent of requested drones required to run the graph. All graphs must have at least 1 drone to run regardless of this setting.
- **ls.brain.controller.droneAllocationTimeout** (integer, default=0) – The number of seconds allowed to collect all drones that the graph requires. Once the controller has all the drones that it needs, it runs the graph. If it fails to get enough drones within this time limit, the graph is not run. A value of zero means that the graph is not run if there are not enough drones immediately available.

4. Automatic restarting of graphs

Certain errors trigger automatic graph restarts. A restart, in this case, means the whole graph restarts as it would if `brainController.py` was executed a second time. Nodes that completed successfully would still run again. The following properties govern this behavior:

- **ls.brain.controller.restartMaximum** (integer, default=0) – The maximum number of times a graph can restart. A value of 0 means a graph will not restart.
- **ls.brain.controller.restartableErrors** (string, default=match any error) – A regular expression (defined by java) that when matched in the error message will restart the entire graph. This item only restarts the maximum number of times as defined by **ls.brain.controller.restartMaximum**. Therefore, if **restartMaximum** is 0 or not defined, then the graph is never restarted no matter what the value of **restartableErrors**.

Website: www.lavastorm.com
Support Email: support@lavastorm.com

Document ID: LAE-TL-1
Date of Publication: Wednesday, October 12, 2016