

## Fine-tuning Bulk Operation Processing

Date published: 2019-04-15

Date modified: 2022-07-21



# Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

# Contents

<b>Fine-tuning bulk operation processing.....</b>	<b>4</b>
Scenario: Frequency and size of batch operation.....	4
Scenario: Heartbeat interval and timeout rate.....	5
Scenario: Heartbeat interval for agent class.....	6
Scenario: Deployed state check frequency and timeout.....	7
 <b>Recommendations for bulk operations in EFM.....</b>	 <b>8</b>

## Fine-tuning bulk operation processing

Learn how to configure the Edge Flow Manager (EFM) bulk operation updates. You can find the related configuration in the `efm.properties` file located in the `conf` directory of the EFM installation.

EFM sends updates to agents through operations. The operations can be divided in the following types:

- Agent level: This is for one individual agent. For example, the Debug request.
- Agent class level: This is for all agents under an agent class. For example, flow update, asset download, property update, and so on.

This documentation describes different configuration scenarios to optimize the agent class level updates.

In order to avoid heavy network loads and prevent Distributed Denial-of-Service (DDoS)-like behavior from agents, the EFM sends updates to agents in smaller batches.

You can find all operation update related configurable parameters in the following list:

```
# The maximum number of operations queued to be sent at once
efm.operation.monitoring.rollingBatchOperationsSize=100

# The interval for refreshing the operation queue
efm.operation.monitoring.rollingBatchOperationsFrequency=10s

# The multiplier for agent heartbeat interval properties
# to calculate timeout for queued operation
efm.operation.monitoring.inQueuedStateTimeoutHeartbeatRate=1.0

# The timeout for deployed, but not done operations
efm.operation.monitoring.inDeployedStateTimeout=5m

# The interval for checking operation timeouts
efm.operation.monitoring.inDeployedStateCheckFrequency=1m
# Enable/disable operation timeouts
efm.operation.monitoring.enabled=true

# Agent heartbeat interval monitor properties
# Default value
efm.monitor.maxHeartbeatInterval=5m

# Class specific values
efm.monitor.maxHeartbeatIntervalForClass.[Class\ A]=1m
efm.monitor.maxHeartbeatIntervalForClass.[Class\ B]=2m
efm.monitor.maxHeartbeatIntervalForClass.[Class\ C]=0m
```

Let us go through the scenarios one by one and see what can change a given parameter in the example deployment with 10000 agents.

### Scenario: Frequency and size of batch operation

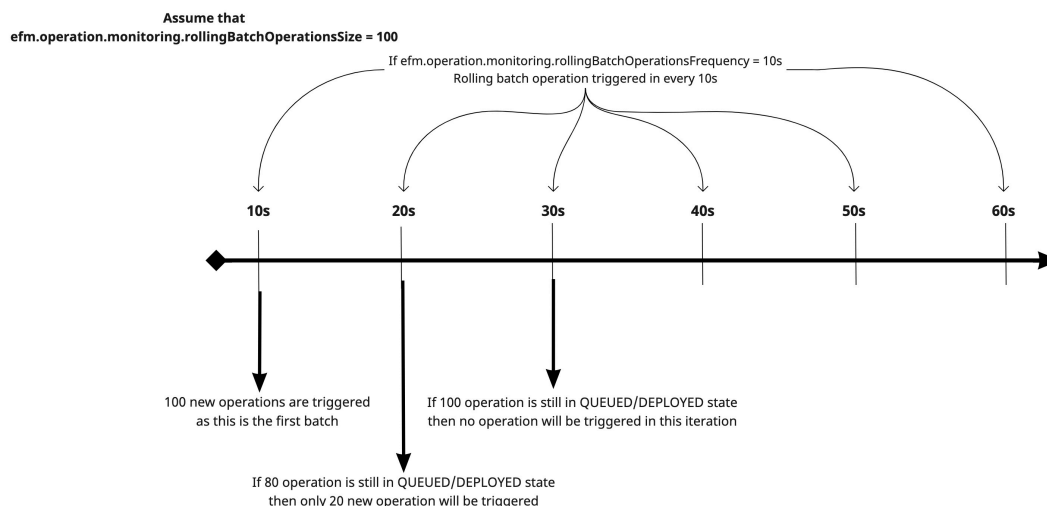
Learn how you can control the frequency of triggering a new batch operation, the number of agents that should receive the new operation, and resolve the issue of network saturation.

#### Scenario

You have 10000 agents and the operation is to download a 100 MB asset to each agent from EFM. The expectation is that an average agent is able to download that asset in 20-30 sec (3-5MB/s). Assume that EFM is able to deal with 100MB/s network IO for this purpose. So, you can see that roughly 20-30 parallel operations can be handled by EFM. This could easily cause network saturation.

## Analysis

On the following diagram you can see with the `rollingBatchOperationsFrequency` parameter you can define how often EFM tries to trigger a new batch operation, and the `rollingBatchOperationsSize` parameter defines how many agents should receive the new operation.



**Important:** If you increase the value of the `rollingBatchOperationsFrequency` parameter, it slows down trigger and mitigate load on EFM. The danger is, too high frequency does not allow enough time for agents to finish with unnecessary triggers.



**Important:** If you increase the value of the `rollingBatchOperationsSize` parameter, it speeds up execution by more agent operations at the same time. The danger is, too large a value causes too much load on EFM.

## Solution

A good starting point could be `rollingBatchOperationsSize=20` and `rollingBatchOperationsFrequency=5s`. In the first 1-4 iterations, the queue becomes full but after it diverges, the operation behaves as expected. The expectation is that roughly 25% of the queue will be free in each iteration with 5s frequency.



### Important:

- If you try to execute this operation without roll batch operation feature, it would cause 10K\*4MB/s network load on EFM.
- Even with a bit higher batch size (for example, 40), EFM easily can run out of new network IO and you experience a DDoS-like load on EFM side.

## Scenario: Heartbeat interval and timeout rate

Learn how you can control the interval between heartbeats and EFM timing out the operation, and resolve the issue of unhealthy agents.

### Scenario

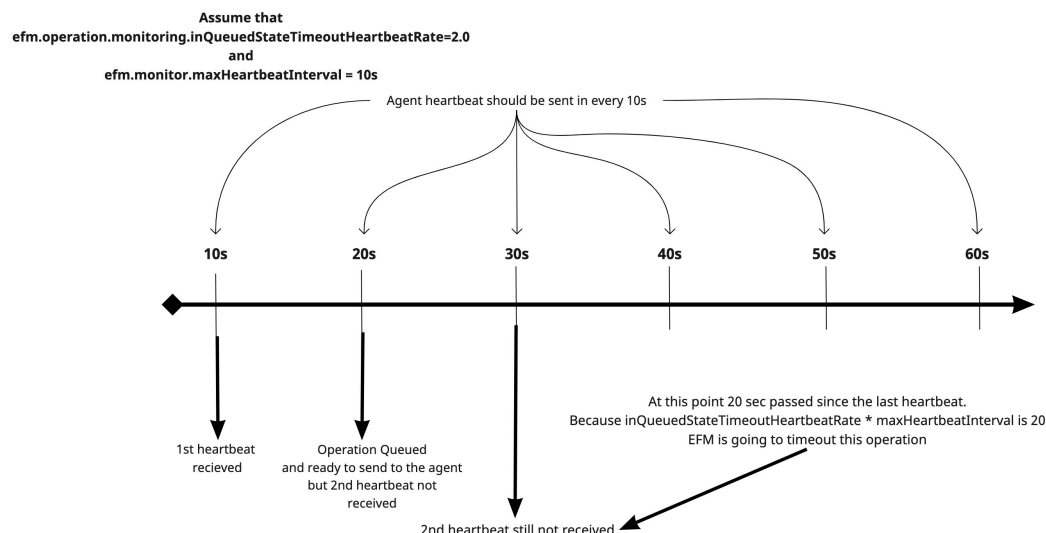
You are running a bulk operation and some of your agents have unstable network access. When the router goes down, you need 1 minute to recover. You know that agents are configured with 10s heartbeat interval.

## Analysis

In case when not all agents are healthy you are faced with the issue that your bulk operation slows down drastically because of the lack of responses from agents. So the operation is stuck at EFM side and unable to send anything to the agent. In order to control this situation, you can define `efm.moni`

`tor.maxHeartbeatInterval=10s` and `efm.operation.monitoring.inQueuedStateTimeoutHeartbeatRate=2.0`.

The following chart shows you how this situation looks like on a timeline:



**Important:** If you increase the value of the `maxHeartbeatInterval` parameter, it allows more time to the agents to give response. The danger is, too large a value expands total execution time.



**Important:** If you increase the value of the `inQueuedStateTimeoutHeartbeatRate` parameter, it allows agents more iteration to come back online. The danger is, EFM should wait unreasonably long for an agent and allow less new operations to be triggered during that time.

### Solution

You can set `maxHeartbeatInterval=12` because if the agent is online you must receive a heartbeat in 10s but you allow some buffer. The 1 min outage could cause you to miss 6-7 heartbeats. So you set `inQueuedStateTimeoutHeartbeatRate=8`. It means EFM waits for 96 seconds before timing out that operation.



**Important:** If you set timeout rate to 9 instead of 8 that would cause ~10% longer execution time for those agents where the operation is timed out.

## Scenario: Heartbeat interval for agent class

Learn how you can control the the heartbeat interval in agent class level and resolve issues.

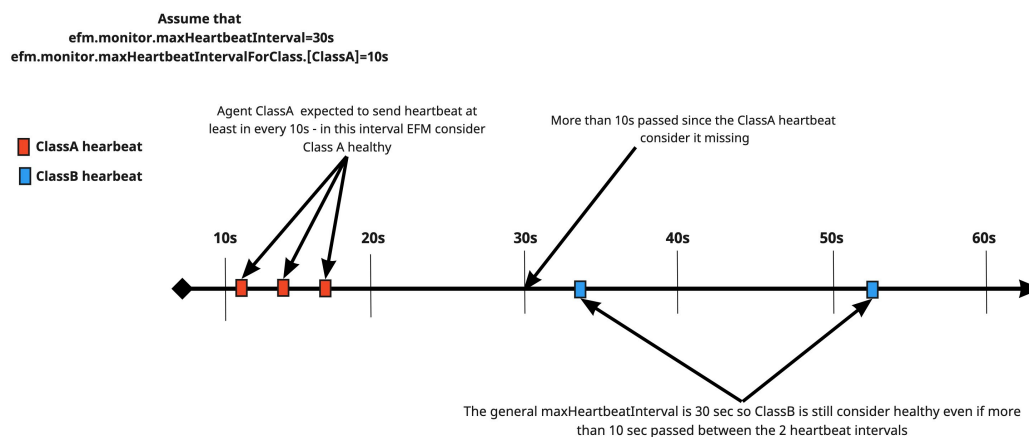
### Scenario

There is an agent class which contains agents deployed in the new department where all the hardware and network performance are beyond the rest of the system and heartbeat is configured to 5s on those agents.

### Analysis

Max heartbeat interval can be configured on class level by using the following parameter: `efm.monitor.maxHeartbeatIntervalForClass.[ClassA]=10s`

The following chart shows you how EFM considers an agent healthy based on these settings:



**Important:** If you increase the value of the `maxHeartbeatIntervalForClass` parameter, EFM allows more time to agents in a specific class. Too many unhealthy agents in this class would slow down bulk operation significantly.

### Solution

So, you can introduce stricter expectations. You set `efm.monitor.maxHeartbeatIntervalForClass.[ClassA]=6s`. So, with `inQueuedStateTimeoutHeartbeatRate=8`, EFM waits only 48 seconds before timing out an operation and notifies you about the issue.

## Scenario: Deployed state check frequency and timeout

Learn how you can control the frequency of checking the state of deployed operations and the timeout value for the deployed operations, and resolve issues.

### Scenario

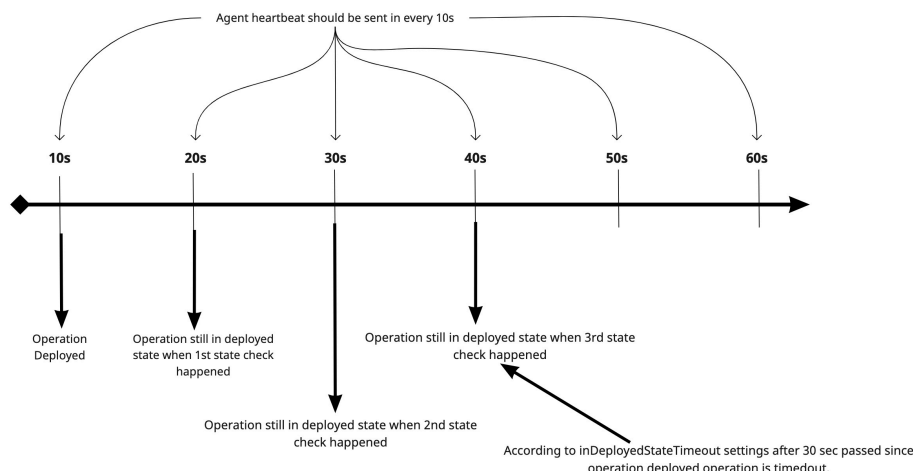
A subset of agents have to deal with other non-CEM applications that require significant percentage of network IO on those agents. This limits the bandwidth available for the subset of agents compared to other agents. As a result, file download time is doubled compared to the expected 20-30 seconds.

### Analysis

Let us assume that agents are healthy but the operation itself is time consuming. In this case EFM is able to send an operation to multiple agents but the agents spend more time executing the operation so that operation remains in deployed state longer. In order to control this situation you can utilize the `efm.operation.monitoring.inDeployedStateTimeout` and `efm.operation.monitoring.inDeployedStateCheckFrequency` parameters.

The following chart gives an overview of how it behaves:

Assume that `efm.operation.monitoring.inDeployedStateTimeout=30s`  
and  
`efm.operation.monitoring.inDeployedStateCheckFrequency=10s`



**Important:** If you increase the value of the `inDeployedStateTimeout` parameter, EFM allows more time to execute an operation. The danger is, even though operation is already done EFM still does not check state and waste execution time.



**Important:** If you increase the value of the `inDeployedStateCheckFrequency` parameter, EFM checks state more frequently to reduce wasted time. The danger is, too frequent checks causes long running deployments.

### Solution

In this case you set `inDeployedStateTimeout=70` and `inDeployedStateCheckFrequency=5s`. So EFM allows enough time for long running file downloads and checks frequently enough to avoid wasting too much time.

## Recommendations for bulk operations in EFM

Learn about the guidances that Cloudera provides to set up the properties in Edge Flow Manager (EFM) for operations.

Cloudera recommends the following while setting up EFM properties:

- `efm.operation.monitoring.rollingBatchOperationsSize`: Set to 10-20% of the total number of agents; but it should not exceed 1000.
- `efm.operation.monitoring.rollingBatchOperationsFrequency`: Based on former execution times, find the frequency where at most 25% of the rolling batch size frees up in a single iteration.
- `efm.monitor.maxHeartbeatInterval` in combination with `efm.operation.monitoring.inQueuedStateTimeoutHeartbeatRate`: Maxheartbeatrate should be close to 75 percentile so you can keep `inQueuedStateTimeoutHeartbeatRate` to a value which should not be more than 3. If a higher number is needed for the rate, you should investigate those agents why those agents do not match with criterias.
- `efm.operation.monitoring.inDeployedStateTimeout` in combination with `efm.operation.monitoring.inDeployedStateCheckFrequency`: Deployed state timeout should be 120% of the longest expected operation execution time. State check frequency should be set to such a value where EFM checks state at most 4-10 times during the operation execution.



- If you put together all the configurations explained in the scenarios, you can get an expected execution time formula:

$$\text{Expected Execution Time} = \overbrace{\frac{\text{numberOfOnlineAgents}}{\text{rollingBatchOperationsSize}} * \text{heartbeatPeriod}}^{\text{Healthy Agents impact on execution time}} + \overbrace{\frac{\text{numberOfMissingAgents}}{\text{rollingBatchOperationsSize}} * \text{inQueuedStateTimeoutHeartbeatRate} * \text{maxHeartbeatInterval[ForClass]}}^{\text{Unhealthy Agents impact on execution time}}$$