|  |  |  |
| --- | --- | --- |
| Huawei Technologies Co. Ltd. | Product version | Confidentiality level |
| C70 | Classified |
| Product name:  FusionInsight | Total pages： |

Superior Planner POC Document

For internal use only

|  |  |  |  |
| --- | --- | --- | --- |
| Authored by | Sam Sanjabi, Du Mike, Lin Ruolin | Date | 2017-03-07 |
| Reviewed by | Chong Chen, Wang GuangMing,  Zhang Wenhu,  Lin XiaoMing | Date | 2017-03-07 |
| Executed by | Lin Ruolin, Du Mike, Lin XiaoMing | Date | 2017-03-07 |
| Resulted by | Lin Ruolin, Du Mike | Date | 2017-03-07 |



Huawei Technologies Co., Ltd.

All rights reserved

Contents

[Superior Planner POC Document 1](#_Toc476663629)

[1. Overview 4](#_Toc476663630)

[2. Environment 4](#_Toc476663631)

[2.1 Cluster Environment 4](#_Toc476663632)

[2.2 Software Configuration 6](#_Toc476663633)

[2.2.1 Core Hadoop 6](#_Toc476663634)

[2.2.2 Hive / HiveServer2 7](#_Toc476663635)

[2.2.3 Superior Scheduler 8](#_Toc476663636)

[2.2.4 TCC 8](#_Toc476663637)

[2.3 New Component Configurations 9](#_Toc476663638)

[2.3.1 Superior Planner 9](#_Toc476663639)

[2.3.2 TCC Adapter 10](#_Toc476663640)

[2.4 TCC Integration with Superior Planner 11](#_Toc476663641)

[2.4.1 tcc support orchestrator action id 11](#_Toc476663642)

[2.4.2 exec-hive.sh script 13](#_Toc476663643)

[3 Workload 14](#_Toc476663644)

[3.1 TCC Workflow (HiCloud) 14](#_Toc476663645)

[3.2 Generating Data 14](#_Toc476663646)

[3.3 Running YARN NoiseMaker 15](#_Toc476663647)

[3.4 Running HiCloud Workflow 16](#_Toc476663648)

[4 Generating Prediction Results 17](#_Toc476663649)

[5 PoC Test Cases 17](#_Toc476663650)

[5.1 Case 1 Single workflow with SLA deadline 18](#_Toc476663651)

[5.1.1 Case Description 18](#_Toc476663652)

[5.1.2 Experiment Steps 19](#_Toc476663653)

[5.1.3 Results (Passed) 19](#_Toc476663654)

[5.2 Case 2 Multiple workflows with different SLA deadline 22](#_Toc476663655)

[5.2.1 Case Description 22](#_Toc476663656)

[5.2.2 Experiment Steps 23](#_Toc476663657)

[5.2.3 Results (Passed) 23](#_Toc476663658)

[5.3 Case 3 Single SLA with reduced Capacity 25](#_Toc476663659)

[5.3.1 Case Description 25](#_Toc476663660)

[5.3.2 Experiment Steps 25](#_Toc476663661)

[5.3.3 Results (Passed) 26](#_Toc476663662)

[5.4 Case 4 Single SLA with application failure/retry 28](#_Toc476663663)

[5.4.1 Case Description 28](#_Toc476663664)

[5.4.2 Experiment Steps 28](#_Toc476663665)

[5.4.3 Results (Passed) 29](#_Toc476663666)

[5.5 Case 5 Single SLA with manual stopped TCC tasks 30](#_Toc476663667)

[5.5.1 Case Description 30](#_Toc476663668)

[5.5.2 Experiment Steps 30](#_Toc476663669)

[5.5.3 Results (Passed) 31](#_Toc476663670)

[6. POC Conclusion 33](#_Toc476663671)

[Appendix 34](#_Toc476663672)

[A.1 Miscellaneous (Optional) 34](#_Toc476663673)

[A.2 Resetting System Components 34](#_Toc476663674)

[A.3 How to check TCC workflow duration 34](#_Toc476663675)

[A.4 How to run graph generation tool to show resource utilization result 36](#_Toc476663676)

[A.5 Canada Env Test Results 37](#_Toc476663677)

[A.5.1 Case 1 Single workflow with SLA deadline 37](#_Toc476663678)

[A.5.1.1 Single SLA: No Planner, Quiet Graph 38](#_Toc476663679)

[A.5.1.2 Single SLA: No Planner, Noisy Graph 39](#_Toc476663680)

[A.5.1.3 Single SLA: Planner, Noisy Graph 40](#_Toc476663681)

[A.5.2 Case 2 Multiple workflows with different SLA deadline 40](#_Toc476663682)

[A.5.2.1 Multiple SLAs without Planner Graph 41](#_Toc476663683)

[A.5.2.2 Multiple SLAs with Planner Graph 42](#_Toc476663684)

[A.5.3 Case 3 Single SLA with reduced Capacity 42](#_Toc476663685)

[A.5.3.1 Single SLA without Planner with reduced capacity Graph 43](#_Toc476663686)

[A.5.3.2 Single SLA with Planner and reduced capacity Graph 43](#_Toc476663687)

[A.5.4 Case 4 Single SLA with application failure/retry 44](#_Toc476663688)

[A.5.5 Case 5 Single SLA with manual stopped TCC tasks 45](#_Toc476663689)

Superior Planner POC

TCC HiCloud Workflow

# Overview

Consumer Business Group uses TCC system and FusionInsight product to manage and run their entire business workflow. Currently, they have challenge to ensure critical business workflows to complete on time. IT CRC team research project (Superior SLA system) is designed to help solving this challenge.

In order to verify the project effect, CBG and IT CRC team agree to conduct a POC test based on subset of CBG workflows and dataset. To support this effort, CBG team enhanced their TCC system to integrate with Superior SLA system and both team run the POC tests.

There are 5 cases which are designed to certify how to guarantee resource for critical job under resource contention, and how to handle most TCC usage scenarios including job retry, cluster capacity change, Tcc task manual maintenance.

**Key Items:**

* DC3.0 Scenario with TCC HiCloud business workflow
* FusionInsight C60 Hadoop Cluster with total 200vcores, 800GB resource, 4 worker nodes
* HiCloud business raw data size is 38GB

# Environment

## Cluster Environment



8.5.132.10 (TCC)

* TCC Orchestrator, MySql Database
* 192GB RAM, 32 Core 2.3GHz

8.5.132.11 (FI Cluster)

* Superior Planner & Adapter, ResourceManager (Superior Scheduler), TCC Gateway, NodeManager, DataNode, Active NameNode
* 256GB RAM, 32 Core 2.3GHz
* NodeManager Size : 200GB memory, 50 vcores
* 3 x 3.6T Disk Partitions used for DN blocks

8.5.131.13 (FI Cluster)

* NodeManager, DataNode, Standby NameNode,
* 256GB RAM, 32 Core 2.3GHz
* NodeManager Size : 200GB memory, 50 vcores
* 3 x 3.6T Disk Partitions used for DN blocks

8.5.131.14 (FI Cluster)

* NodeManager, DataNode
* 256GB RAM, 32 Core 2.3GHz
* NodeManager Size : 200GB memory, 50 vcores
* 3 x 3.6T Disk Partitions used for DN blocks

8.5.131.15 (FI Cluster)

* NodeManager, DataNode, Hive Server
* 256GB RAM, 32 Core 2.3GHz
* NodeManager Size : 200GB memory, 50 vcores
* 3 x 3.6T Disk Partitions used for DN blocks

**Software Versions**

* FusionInsight Hadoop V100R002C60SPC200
* TCC (with orchestrator id added modification)
* FI Hive 1.3.0 (with MR task hook modification)
* FI Spark 1.5.1

## 2.2 Software Configuration

### 2.2.1 Core Hadoop

#### Standard Configurations

* Superior scheduler enabled
* Resource Manager recovery disabled
* Node Manager shuffle service enabled
* CGroups support enabled
* Log Aggregation + job history server enabled
* ACL disabled

#### Container Memory

* mapreduce.map.memory.mb = 4096
* mapreduce.map.java.opts = -Xmx3276M
* mapreduce.reduce.memory.mb = 8192
* mapreduce.reduce.java.opts = -Xmx6553M
* yarn.app.mapreduce.am.resource.mb = 2048
* yarn.app.mapredcue.am.command-opts = -Xmx1638M

Notes:

* Reducer and AM memory setting is different than CBG setting.
* With fake data, we found task 4093019 need more reduce memory otherwise huge I/O of one reduce takes unreasonable amount of time, so we upped the reduce memory for all tasks.
* CBG AM setting that we see makes no sense: they set AM container request to 4096M but JVM size is only 1024M, too much buffer.

#### Resource Overcommit

* yarn.nodemanager.resource.cpu-vcores = 80
* yarn.nodemanager.resource.memory-mb = 327680

#### Slow Start and Shuffle Buffer

* mapreduce.job.reduce.slowstart.completedmaps = 0.9
* mapreduce.reduce.shuffle.input.buffer.percent = 0.2

Notes:

* slowstart was set this way to copy CBG setting for Hive jobs
* shuffle input buffer setting was changed because with the increased reducer memory, the default setting caused task 4093013 to fail in the last stage.

### 2.2.2 Hive / HiveServer2

#### SQL Authorization

This was enabled to get the TCC hive scripts to work unmodified (e.g. ‘set role admin’)

* hive.users.in.admin.role = root,omms
* hive.security.authorization.enabled = true
* hive.security.authorization.manager = org.apache.hadoop.hive.ql.security.authorization.plugin.sqlstd.SQLStdHiveAuthorizerFactory

Additionally, need to launch HiveServer2 with the last 3 parameters above passed to it using –hiveconf option to get it to work. FI environment already has SQL Authorization enabled, you don’t need to worry about this stuff.

#### Disable Auto Map Join

I was getting many errors in MapJoin tasks in my env, particularly with the custom CBG UDFs, so I disabled them.

* hive.auto.convert.join = false

#### Hive Hook Configuration

To install the hive hook, follow the instructions in the “Updating Hive Installation” section of the README at $SUPERIOR\_PLANNER\_HOME/examples/README. Below is instructions just to configure the parameters:

First we need to enable hive to allow runtime modification of orchestrator action id etc… when submitting from client:

* hive.security.authorization.sqlstd.confwhitelist.append = splanner\..\*

Configure Hive to load the hook JAR file:

* hive.aux.jars.path = file:///path/to/planner/lib/superior-planner-queuename-handler-1.0.0.jar

Set the hook to run the cleanup task every hour

* splanner.hive-hook.record.cleanup.interval.mins = 60

Set the hook to delete old action/session/query records from memory after they are inactive for 12h:

* splanner.hive-hook.record.expirty.time.mins = 720

Configure the reservation-enabled queues that the hook should modify the queue name for

* splanner.hive-hook.reservation.queues = root.Plan

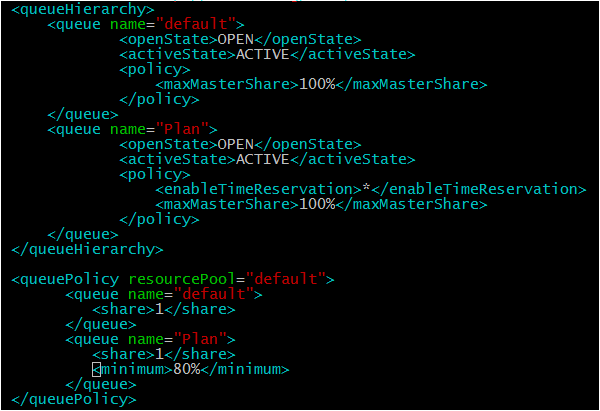
Configure hive hook console logs (optional):

* splanner.hive-hook.log.to.console = true

### 2.2.3 Superior Scheduler

#### superior-queues.xml

Configure at least one reservation-enabled queue owning the desired resources. Below it is queue named Plan which owns 80% the cluster resources:



#### superior-resource-pools.xml

Configure default pool containing all resources:



#### superior-scheduler.xml

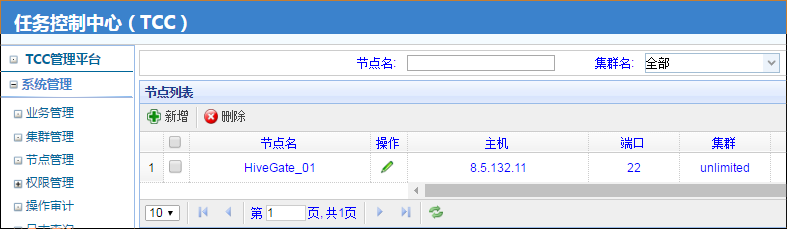
To replicate our results, disable accessUnreservedResource.



### 2.2.4 TCC

In our environment, we only changed tcc.benchDate in tcc.sysconfig.properties to the earliest day that we generated data for.

In addition we configured our gateway node (8.5.132.11) to have infinite slots as follows:

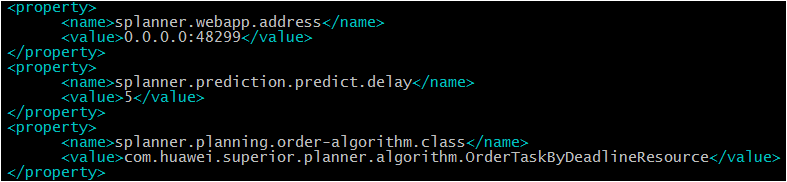
We also increased the timeout of every step in the hicloud\_day flow by a factor of 4 by editing the database.

## 2.3 New Component Configurations

### 2.3.1 Superior Planner

#### superior-planner.xml

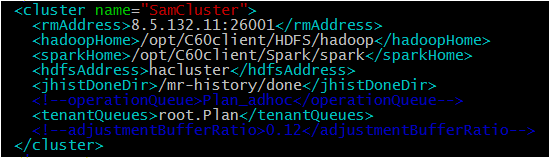
Since we are running the test with infinite gateway, we have no special configuration here to set a limit on gateway resources. The one optional configuration you might want to set is to change the frequency that the predictor runs with splanner.prediction.predict.delay. The following configuration sets this to 5 minutes:



Once one complete set of prediction results have been generated, the predictor can be disabled if you wish. This way any execution errors won’t affect the prediction results.

#### superior-planner-cluster.xml

The following configuration starts the planner with minimal configuration required, with the ‘root.Plan’ as tenant queue for plan creation and default queue as the operation queue for the predictor Spark job.



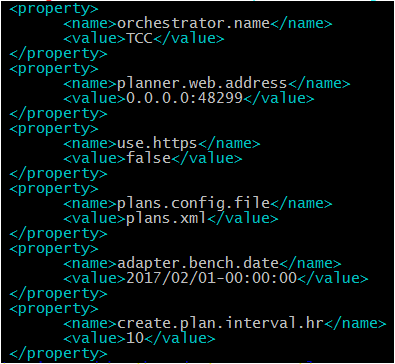
Notes:

* Cluster name is just a string, can be any name. It is used internally by the planner to identify different Hadoop clusters.
* rmAddress must specify RM web listening port
* The order of that the parameters are specified is important, it must be in the above order.
* After one full set of prediction results are generated, sparkHome can be commented out and Planner restarted in order to disable predictor.

### 2.3.2 TCC Adapter

#### orchestrator-adapter.xml

The only important parameter here is to set the adapter.bench.date to at least include the day(s) that have data generated for the test. In our case we generated data for Feb 1st 2017 in our preliminary trials so we set the bench date in both TCC and adapter to this date:



Notes:

* orchestrator.name is just a string, can be any name. It is used internally by the planner to identify different submitting orchestrators.
* planner.web.address should match the setting in superior-planner.xml

## 2.4 TCC Integration with Superior Planner

TCC Job Scheduler system needs to integrate with Superior Planner to guarantee critical workflow SLA. TCC Job Scheduler should set unique orchestrator workload ID into hive query environment, so that Planner can automatically analyze and group independent YARN jobs (Map Reduce) into single TCC task and predict sub-DAG resource usage.

### 2.4.1 tcc support orchestrator action id

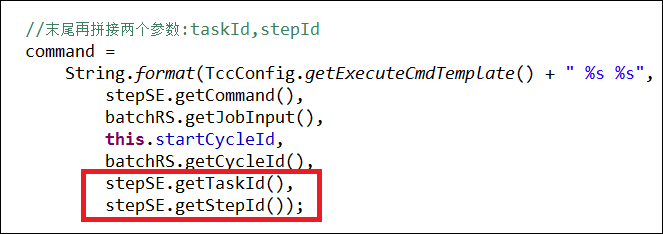
The Superior Planner assumes a particular format for IDs that the workflow orchestrator provides.

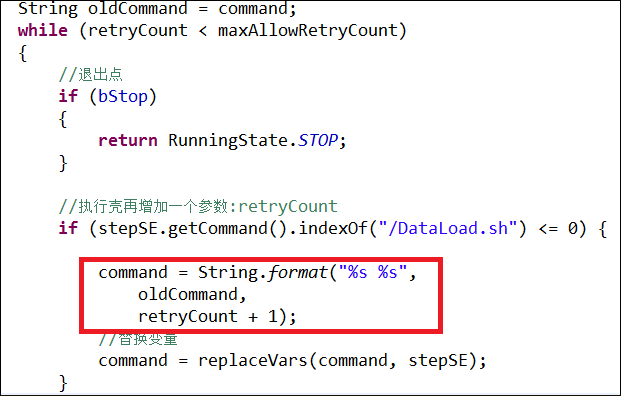
In TCC case, a full orchestrator action ID will be defined as following:

**${TCC\_TASK\_ID}:${TCC\_PERIOD\_DATE}-${TCC\_PERIOD\_HOUR}.${TCC\_STEP\_NUMBER}:1.${TCC\_STEP\_ATTEMPT\_NUM}**

* TCC Orchestrator

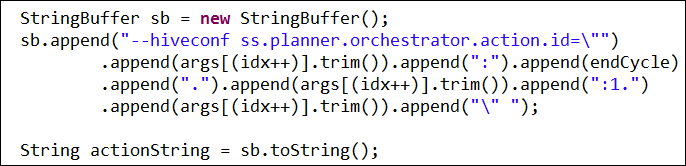
When TCC Scheduler submit task to hive, add taskId, StepId, and retry time info in cmd. The change is only in the CycleTask.java





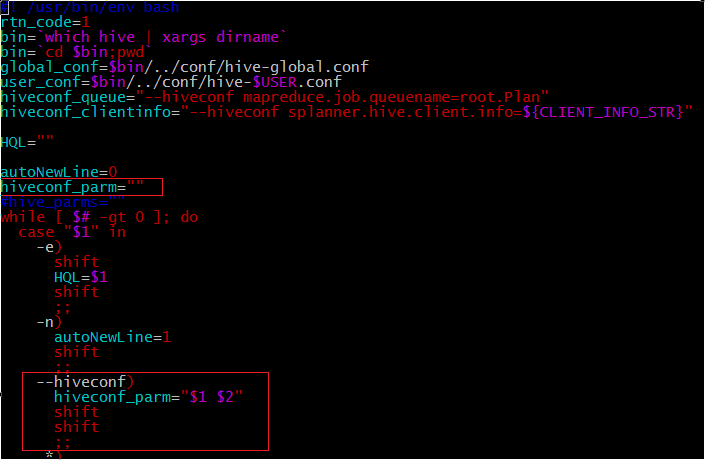
* Hive-ExecutorShell.jar

The Hive ExecutorShell has also added logic to parse the newly added three parameters from TCC Orchestrator and parse to required orchestrator action id to hive beeline.



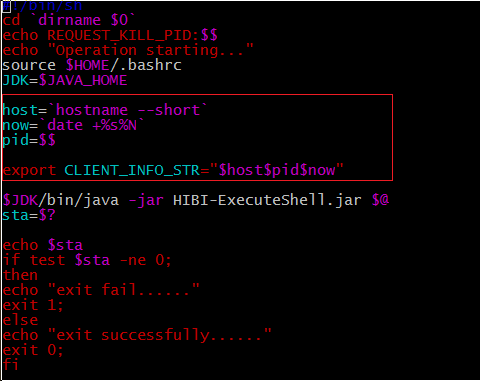
* HIVE SCRIPT

The hive script was updated to add the queue name, user , orchestrator action id, and client info string to the hive submission:



### 2.4.2 exec-hive.sh script

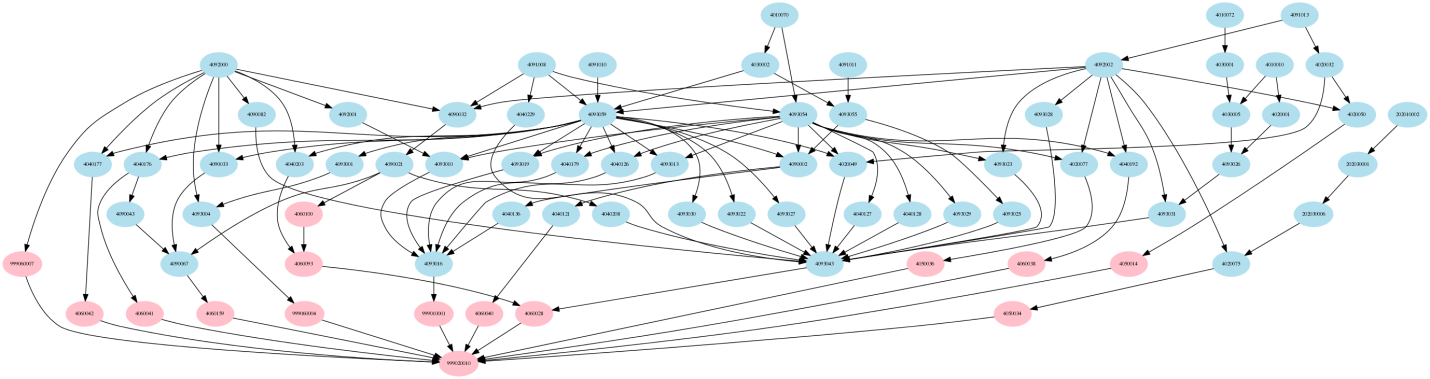
We updated this script to add CLIENT\_INFO\_STR parameters to the execution environment of hive script:



The CLIENT\_INFO\_STR is just appending the hostname, pid, and current timestamp. This is sent to the hive hook to allow smoother operation in POC scenario. In production environment, it shouldn’t be needed.

# Workload

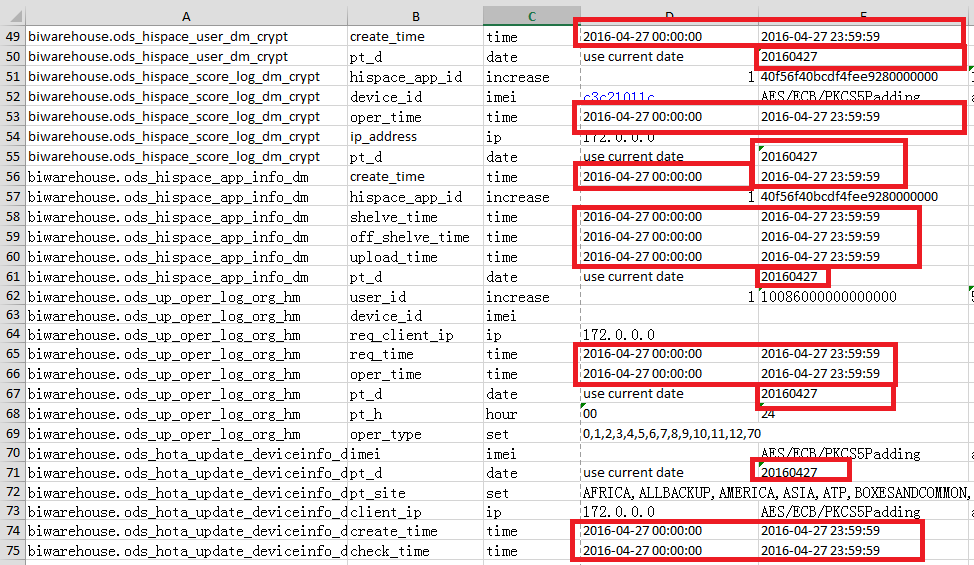
## 3.1 TCC Workflow (HiCloud)



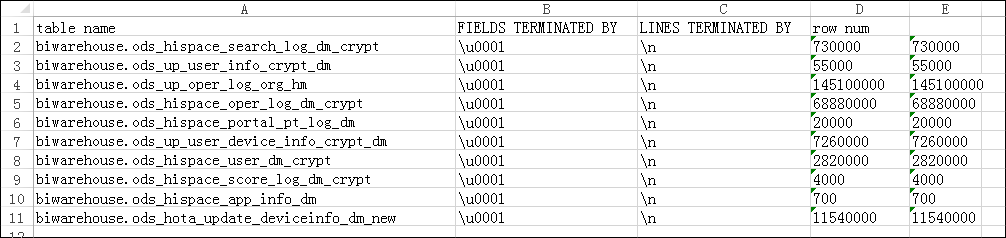
In this POC, we will be using HiCloud workflow. HiCloud workflow includes 75 TCC tasks and 248 map reduce jobs. In regular TCC environment, at the end of run, hiCloud workflow generates a critical report. SLA requirement requires it must complete at 6:00 AM in the morning. So, all HiCloud workflow tasks are critical tasks from Planner perspective. Here is the task dependency graph for all TCC tasks of HiCloud workflow.

## 3.2 Generating Data

As per instructions, we generated data using the hicloud\_ods\_sample.xlsx spreadsheet and setting all the dates in the “create method” sheet to match the date partition we want to generate:



For the “create table” sheet, we left the number of rows same as default values:



**The initial data size was ~38G.**

## 3.3 Running YARN NoiseMaker

In TCC production environment, there are mixes of critical and non-critical workflows competing resources. Since POC environment only has TCC Hicloud tasks, which are all critical workloads, in order to demonstrate resource contention when running critical vs non-critical workloads, we developed a noise maker program (YARN NoiseMaker) to simulate low priority non-critical workloads, which compete cluster resources with critical HiCloud workloads

Its usage is simple, from $SUPERIOR\_PLANNER\_HOME/examples/NoiseMaker directory, run the noisemaker.sh script as follows:

$ ./noisemaker.sh <nclients> <pause\_time> <queue>

The script spawns <nclients> concurrent clients that each alternate between

1. submitting a M/R sleep job with 30 execution containers, with each sleep taking either 1 min (20% chance), 5 minutes (70% chance), or 25 minutes (10% chance)
2. sleeping locally for a random amount of time between 1 and <pause\_time> seconds

Jobs are submitted to the YARN queue specified by the <queue> parameter, which is the only optional parameter. It is possible to alter the workload executed by each job, but at the moment that can only be done by modifying the noise\_client.sh script directly.

For advanced users, the relevant variables in noise\_client.sh to change to customize the workload are:

MR+SLEEP\_CMD

* Command that is executed by each job, you can change the number of containers per job here, which is hard-coded to 30 at the moment.

ACTIONS

* List of actions that each job cycles through. Current valid values are:
  + SLEEP – sleep locally
  + DISTSH\_SLEEP – submit a sleep job through distributed shell (no jobhistory)
  + MR\_SLEEP – submit the M/R sleep job described above (default)
  + PI – submit a M/R Pi job with 30 containers

SLEEP\_DIST/SAMPLES\_DIST

* If action is DISTSH\_SLEEP or PI, these lists define the distribution that the length of the sleep or number of samples of the Pi job is selected from.
* If the action is the default MR\_SLEEP, the distribution is defined by the ‘sleep\_dist’ file in the NoiseMaker directory.

## 3.4 Running HiCloud Workflow

In our tests, our goal was to be able to control the TCC system so that we can run one single day’s instance of the hicloud\_day workflow on a repeated basis so that we can conduct our experiments. Broadly speaking, the way we achieve this is by:

1. Generating the dataset for our target date (in our case 20170201) as described in the “Generating Data” section above.
2. Setting TCC’s “Bench Date” system property to the same date
3. Using a script to set all instances of hicloud\_day tasks after 20170201 to “failed” () status in the database.
4. Using a script to set all 20170201 hicloud\_day tasks to “init” () to init status in the database.

In this way, we can make sure TCC only ever submit’s jobs for 20170201. Furthermore, if we ever want to rerun the trial, we just repeat step 4 above.

# Generating Prediction Results

Before running the PoC test cases with the Planner fully operational, we need to generate a set of prediction results to be used by the planner to generate its reservation allocations. This is called a “cold start” scenario, where the Hadoop cluster and planner history information has no prior information that the predictor can use to inform the planner.

In order to generate prediction results, the steps are:

1. Start the Planner as configured above
2. Start the TCC Adapter in log-only mode
3. Run hicloud\_day workload for your target day using the procedure described above.

If all tasks complete successfully, make sure to observe the Spark prediction job execute one time after all tasks are done, then you can stop the planner + adapter.

To validate that the prediction results are complete, you can verify that the generated prediction results file has exactly 60 records as follows:

hdfs dfs –cat ${splanner.shared.dir}/predictor/results/part-00000 | wc –l

Where the planner shared dir is as configured in superior-planner.xml (the default is /planner).

*Note: By default, planner prediction module will periodically submit spark predication workloads to refresh prediction result based on latest data, which will occupy some cluster resources. For POC case, since we only have limited machines, after completing this task successfully, it is recommended to comment out the sparkHome parameter from the superior-planner-cluster.xml file to prevent the prediction job from executing. This will prevent the predictor from running and allow you to run your experiments without having the predictor update the results.*

# PoC Test Cases

The following table describes the major cases we executed to demonstrate planner’s value. In all cases, all workload is submitted to the same YARN tenant.

|  |  |  |
| --- | --- | --- |
| # | Case Name | Description |
| 1 | Single SLA Workflow | Run a single day of hicloud\_day workload on cluster. In this case we measure:   * Overall execution time of the workflow to see whether we can meet deadline |
| 1.1 | No Planner Quiet | Run a single day of hicloud\_day workload with no additional noise generated |
| 1.2 | No Planner  Noisy | Run a single day of hicloud\_day workload with additional noise generated using YARN NoiseMaker |
| 1.3 | Planner Noisy | Run a single day of hicloud\_day workload with additional noise generated using YARN NoiseMaker, and the Superior II Planner system active. |
| 2 | Multiple SLA | Run 3 days of the workflows simultaneously with staggered SLA. Measure:   * Overall execution time of the each day’s workflow   **NOTE**: in order to generate sufficient contention, this case was executed on a reduced cluster capacity (each with smaller vcores and total memory) with the same configuration as before.  Subcase 0 shows the baseline of one instance of the HiCloud\_Day workflow running on this reduced cluster size. |
| 2.1 | No Planner | Run the case without planner active |
| 2.2 | Planner | Run the case with planner system active |
| 3 | Single SLA with reduced capacity | Run 1 days of the workflows simultaneously with noise. In the middle of run (say 1 hour), reduce system capacity from 4 nodes to 2 nodes to see impact of overall SLA of critical workflow. |
| 3.1 | No planner | Run the case without planner active |
| 3.2 | Planner | Run the case with planner active |
| 4 | Single SLA with application failure/retry | Run 1 days of the workflows simultaneously with noise. During the run, application failed and TCC automatically rerun the program, Planner should be able to catch up and continue plan. |
| 5 | Single SLA with human manually stopped TCC tasks | Run 1 days of the workflows simultaneously with noise. In the middle of run (say 30 minutes), administrator goes in and stops some TCC tasks to simulate the real situation. After 30 minutes, administrator resumes the tasks again. |
|  |  |  |

## 5.1 Case 1 Single workflow with SLA deadline

### 5.1.1 Case Description

In CBG production environment, during mid night, in order to guarantee resources for critical workflow, people normally reduces total number of allowed concurrent TCC tasks to reduce resource contentions for critical workloads. This actually reduces total cluster utilization. And even with this approach, in worse scenario, when contention occurs, it may still make critical workflow run much longer, potentially missing the expected deadline. We created this case to simulate a worse case scenario.

Basically, we will run the TCC hi\_cloud workflow in an empty cluster to get a baseline runtime. Then we will run hi\_cloud critical workflow together with noise programs to simulate concurrent low priority workloads. Both of them are running in the system within single tenant-queue, root.Plan. Without planner, we will expect entire workflow potentially run much longer and beyond deadline. Then we will repeat the same case with Planner, we should be able to see a much shorter finish time of workflow, meanwhile cluster utilization is also high.

### 5.1.2 Experiment Steps

1. Start with all systems turned off (adapter, tcc, planner, rm).
2. Start resource manager, check GUI to check that all 4 nodes have connected, that cluster has 200 vcores, 800GB memory, and that there are no jobs running. Each node should have 50 vcores, 200GB memory.
3. If running planner: turn on planner, check that it is communicating with the yarn cluster, and is reporting that the cluster has 200 vcores, 800GB memory.
4. Set all tasks in TCC database to done.
5. Set day 1 (in this case, it was 20170201) in TCC database to initialized.
6. If running noise: Turn on noise maker script, with 15 instances, 0 delay between runs. Check that the cluster is completely saturated before continuing.
7. If running planner: turn on TCC adapter, check that it reports that the workflow has 75 nodes, and the plan is submitted successfully.
8. Start TCC.
9. Wait for all jobs to finish.
10. Record run time from TCC GUI.
11. If desired: generate graphs to show cluster utilization.
12. If running noise: Turn off noise maker script. Run kill\_the\_noise.sh from another window, then either wait, or kill all of the noise applications in YARN.

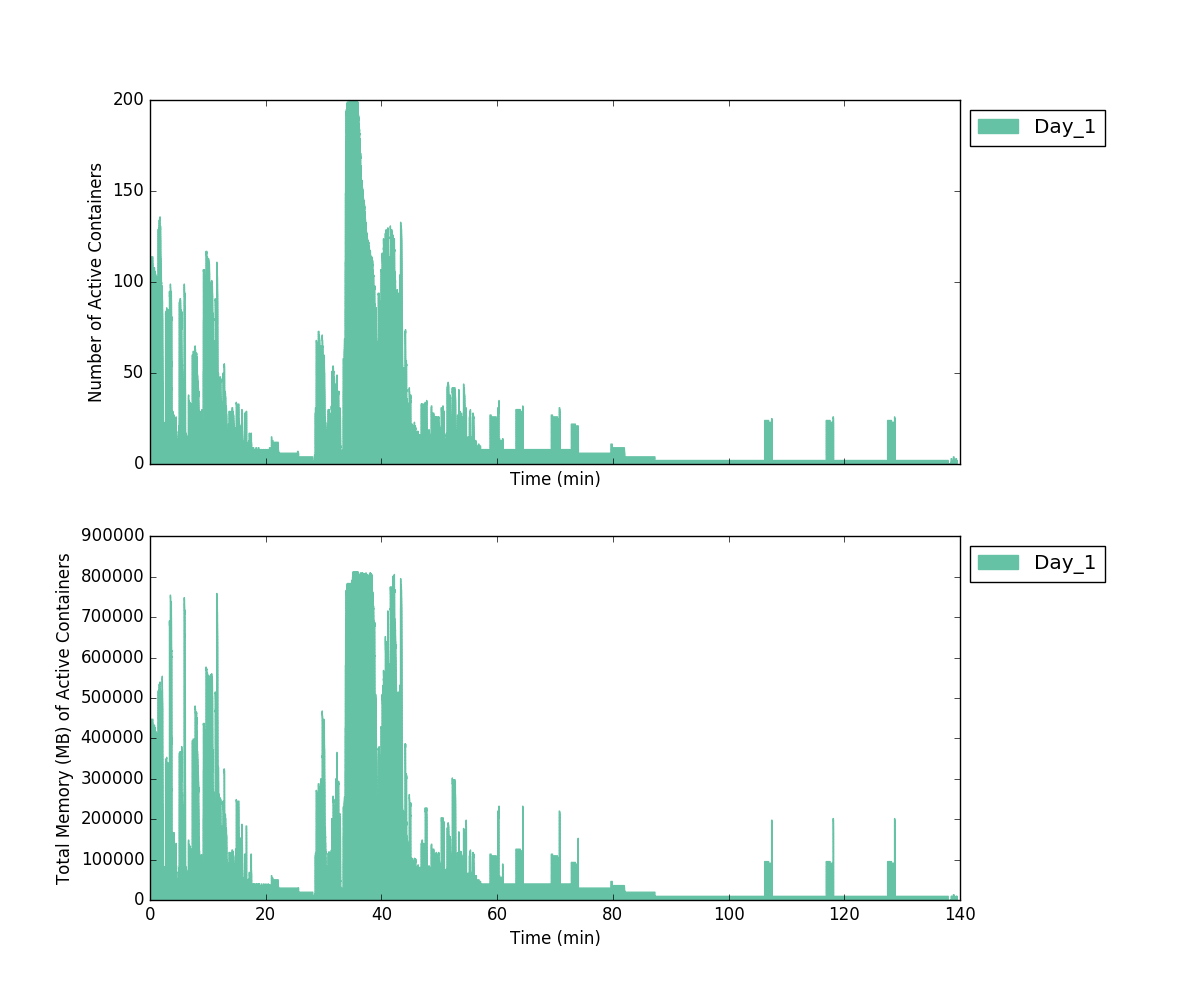
### 5.1.3 Results (Passed)

Check TCC web page to get end to end hi\_cloud duration.

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 1.1 | Single SLA: No Planner, Quiet | hicloud\_day execution time: 2h 20m 7s |
| 1.2 | Single SLA: No Planner, Noisy | hicloud\_day execution time: 4h 41m 27s |
| 1.3 | Single SLA: Planner with Noisy | hicloud\_day execution time: 2h 41m 37s |

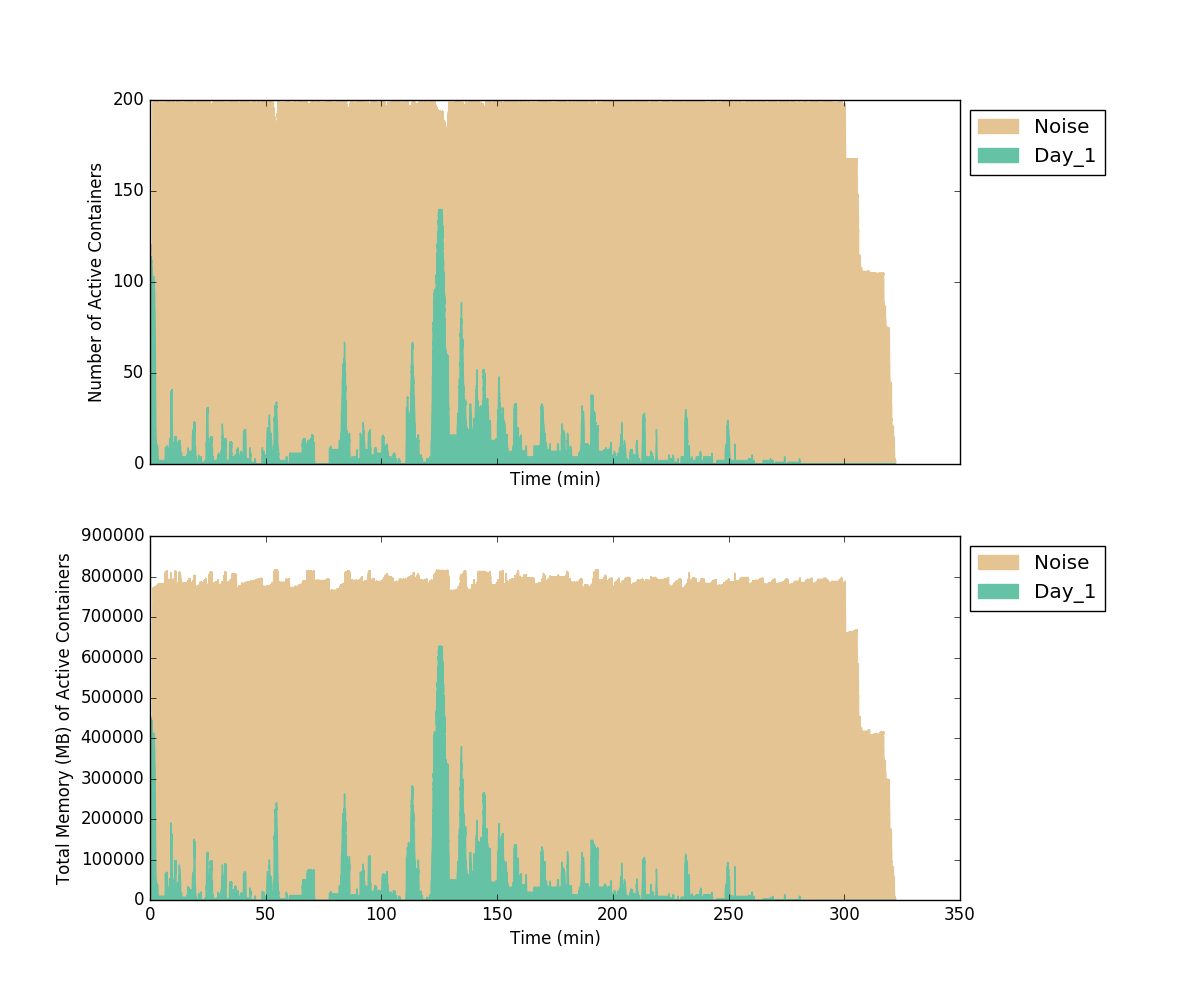
#### 5.1.3.1 Single SLA: No Planner, Quiet Graph

In an empty cluster, entire workflow takes about 2 hours 20 minutes to complete.



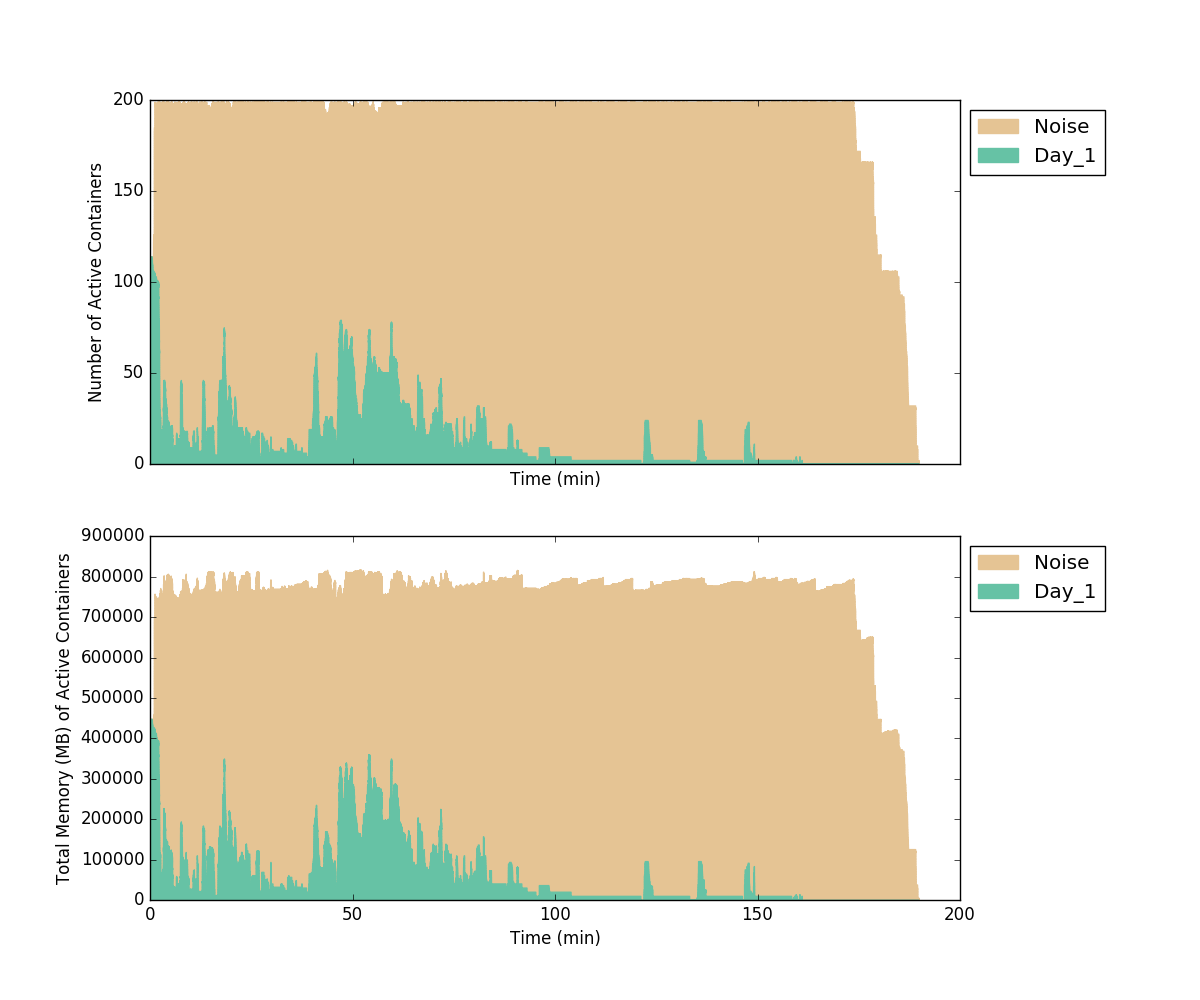
#### 5.1.3.2 Single SLA: No Planner, Noisy Graph

With Noisy in the place, entire workflow takes about 4 hours 41 minutes to complete. If we set deadline as 3 hours, clearly, we are beyond deadline.



#### 5.1.3.3 Single SLA: Planner, Noisy Graph

With Noisy in the place and Superior Planner, entire workflow takes about 2 hours 41 minutes to complete, while still maintaining pretty good overall cluster utilization.



## 5.2 Case 2 Multiple workflows with different SLA deadline

### 5.2.1 Case Description

In some cases, customers may run several independent critical workflows with different SLA deadlines. In current TCC environment, if workflows are independent and all critical, TCC scheduler cannot distinguish them and will try to run all of them at the same time, which will potentially impact workflow with earlier deadline.

In order to simulate this case, we run 3 independent hi\_cloud workflows (Feb 1st 2017, Feb 2nd, 2017 and Feb 3rd, 2017) through TCC and set them with different deadlines. The first case will let them run through the system without planner, we should be able to see first day deadline missing. The second case will run the same workloads with planner, we should be able to see all deadline met.

### 5.2.2 Experiment Steps

Since we only have one hi\_cloud workflow, we will create three days instances of hi\_cloud to simulate independent workflows. To do so, we need to unset “cycle” flag on some jobs to make them run independently.

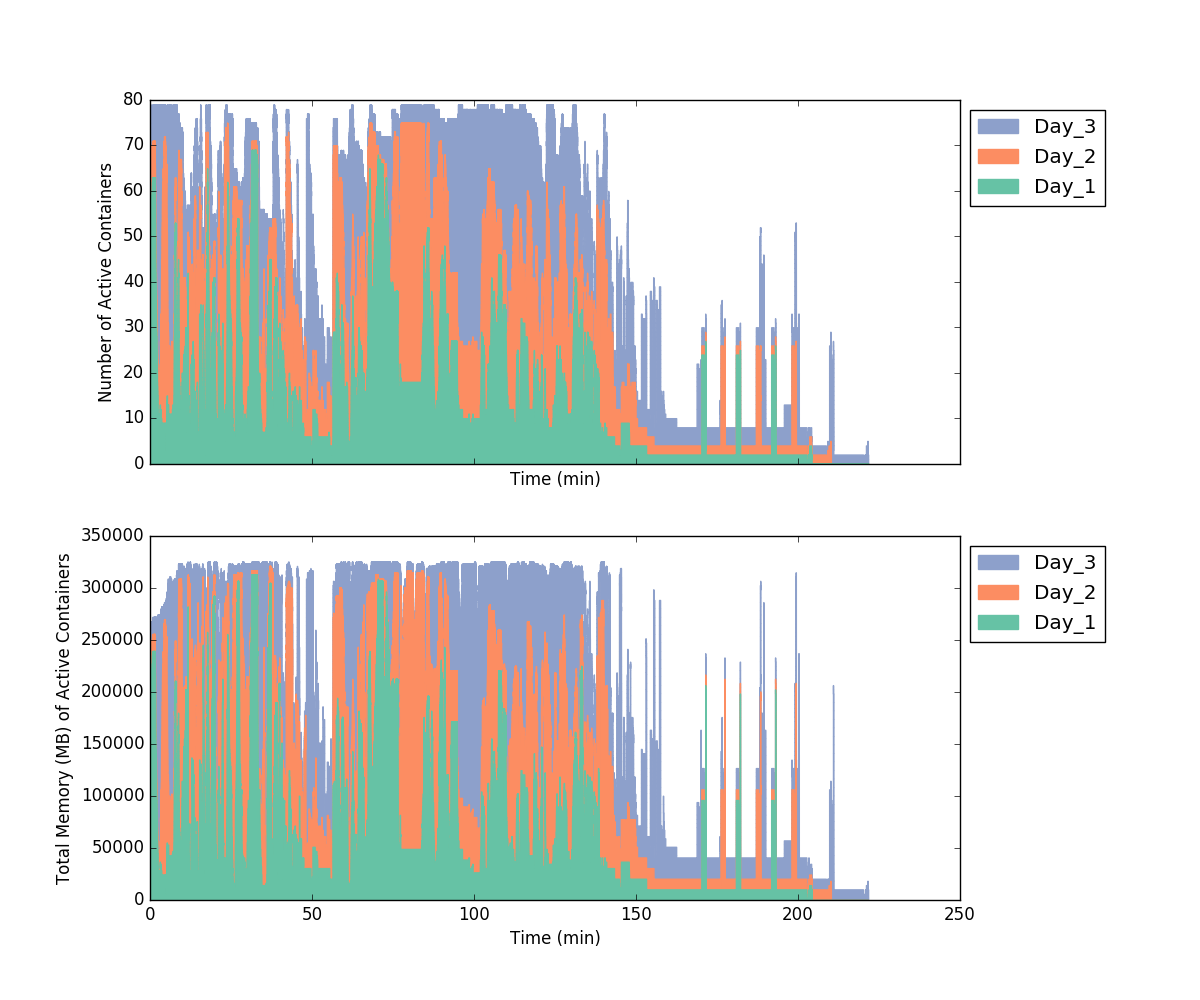
1. Start with all systems turned off (adapter, tcc, planner, rm).
2. In TCC database, for all jobs in hicloud\_day workflow, set Cycle\_Depend\_Flag = 0, except for jobs with Task\_ID 4092002 and 4040136.
3. Start resource manager, check GUI to check that all 4 nodes have connected, that cluster has 80 vcores, 320GB memory, and that there are no jobs running. Each node should have 20 vcores, 80GB memory.
4. If running planner: turn on planner, check that it is communicating with the yarn cluster, and is reporting that the cluster has 80 vcores,320GB memory.
5. Set all tasks in TCC database to done.
6. Set day 1, day 2 and day 3 (in this case, it was 20170201, 20170202, 20170203) in TCC database to initialized.
7. If running planner: turn on TCC adapter, check that it reports that the workflow has 225 nodes, and the plan is submitted successfully.
8. Start TCC.
9. Wait for all jobs to finish.
10. Record run time from TCC GUI.
11. If desired: generate graphs to show cluster utilization.
12. Change back each node to 80 vcores, 320GB memory.
13. If desired, set Cycle\_Depend\_Flag = 1 for those jobs changed in step 2 above.

### 5.2.3 Results (Passed)

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 2.1 | Multiple SLA no Planner | **hicloud\_day 1: 3 hours 24 min 57 seconds**  hicloud\_day 2: 3 hours 30 min 57 seconds  hicloud\_day 3: 3 hours 42 min 17 seconds |
| 2.2 | Multiple SLA with Planner | hicloud\_day 1: 2 hours 40 min 05 seconds  hicloud\_day 2: 3 hours 29 min 25 seconds  hicloud\_day 3: 4 hours 21 min 35 seconds |

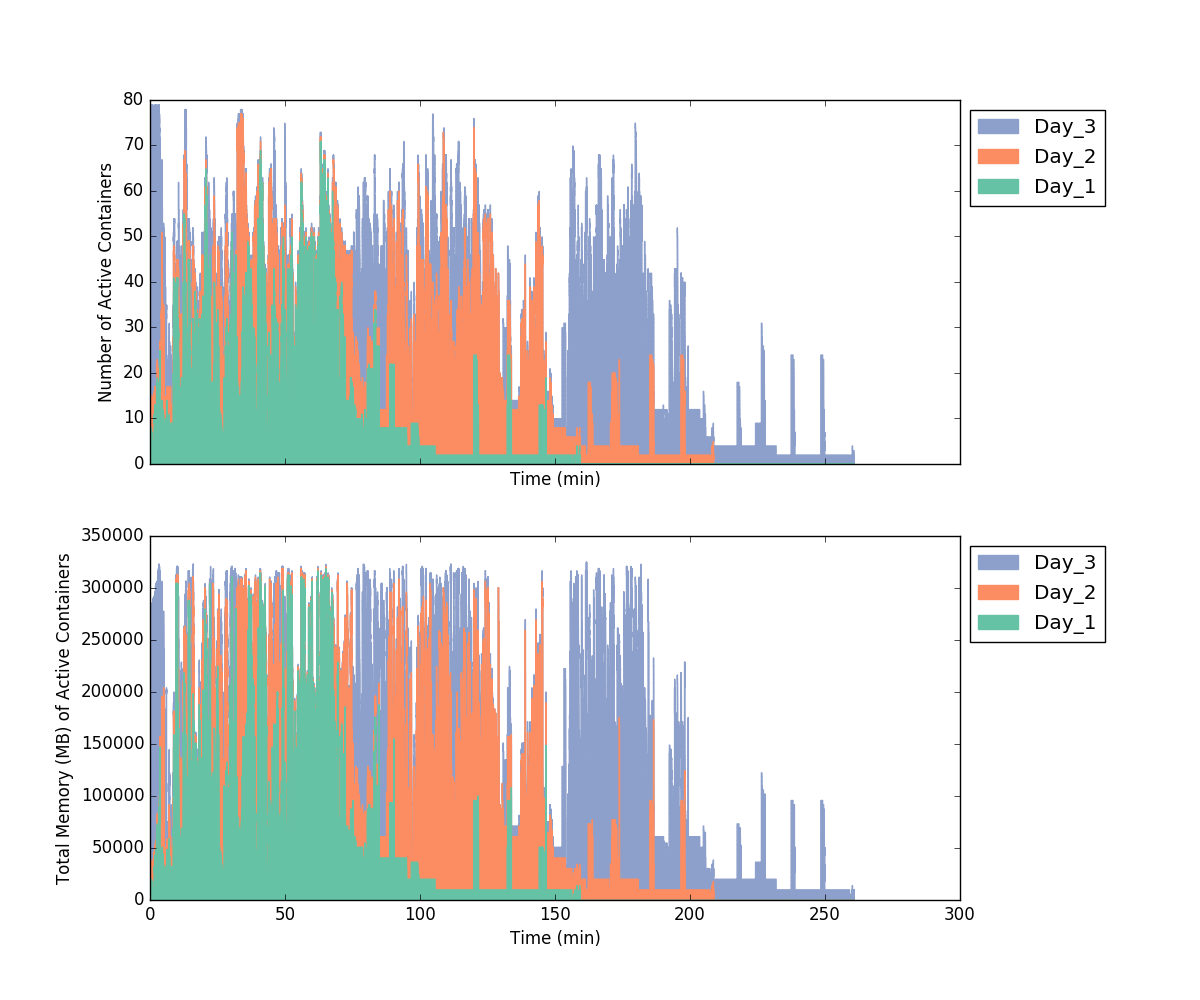
#### 5.2.3.1 Multiple SLAs without Planner Graph

Without Planner, the first day workflow actually runs over deadline.



#### 5.2.3.2 Multiple SLAs with Planner Graph

With planner, all critical jobs are able to complete within their deadlines



## 5.3 Case 3 Single SLA with reduced Capacity

### 5.3.1 Case Description

Cluster capacity may be changed due to some unexpected reasons, for instance, machines power outage or hardware failure etc. When cluster capacity is reduced, it will impact original critical workflow SLAs since workloads compete for less available resources. With planner in place, Superior Planner intelligently identifies critical workloads and guarantees resources for those workloads to make sure to complete those workflows within SLA deadline.

In order to simulate this case, we run hi\_cloud workflows (Feb 1st 2017) through TCC together with noisy workloads in a 4 nodes FI cluster. After one hour run, we will manually shut down 2 of FI nodes to create extreme scenario. We will run this case with and without planner component to see result.

### 5.3.2 Experiment Steps

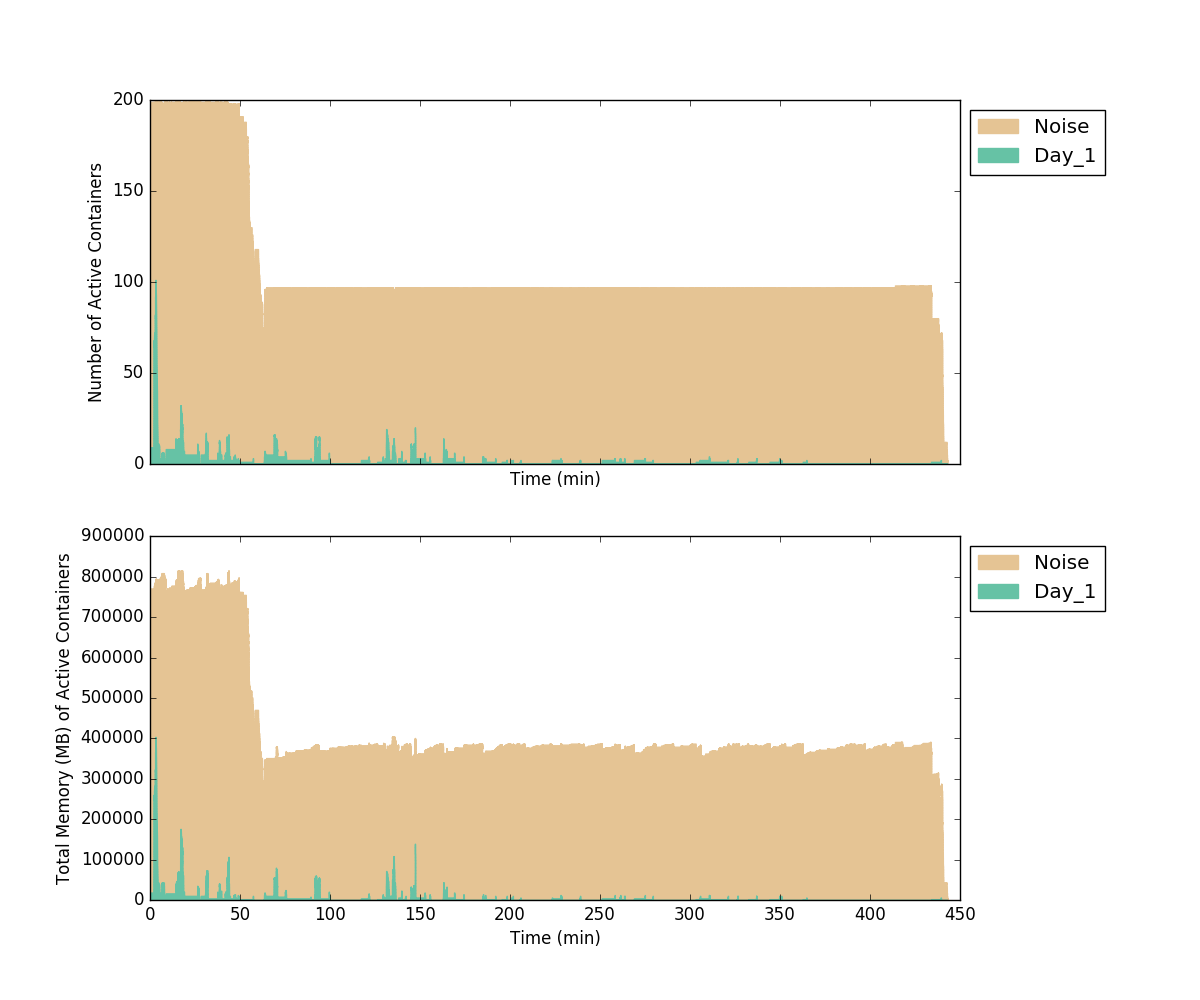
1. Start with all systems turned off (adapter, tcc, planner, rm).
2. Start resource manager, check GUI to check that all 4 nodes have connected, that cluster has 200 vcores, 800GB memory, and that there are no jobs running. Each node should have 50 vcores, 200GB memory.
3. If running planner: turn on planner, check that it is communicating with the yarn cluster, and is reporting that the cluster has 200 vcores, 800GB memory.
4. Set all tasks in TCC database to done.
5. Set day 1 (in this case, it was 20170201) in TCC database to initialized.
6. Turn on noise maker script, with 15 instances, 0 delay between runs. Check that the cluster is completely saturated before continuing.
7. If running planner: turn on TCC adapter, check that it reports that the workflow has 75 nodes, and the plan is submitted successfully.
8. Start TCC.
9. After 1 hour from the start time as reported by the TCC GUI, kill 2 nodes. In this case, we shut down two nodemananger instances (8.5.132.11, 8.5.131.13) by FI portal.
10. Wait for all jobs to finish. If running without planner, it is OK to terminate the test early after 5 hours.
11. Record run time from TCC GUI.
12. If desired: generate graphs to show cluster utilization.
13. Turn the 2 nodes back on.
14. Turn off noise maker script. Run kill\_the\_noise.sh from another window, then either wait, or kill all of the noise applications in YARN.

### 5.3.3 Results (Passed)

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 3.1 | Single SLA no Planner with reduced capacity | hicloud\_day execution time: more than 7 hours, still running |
| 3.2 | Single SLA with Planner and reduced capacity | hicloud\_day execution time: 3 hours 13 min 16 seconds |

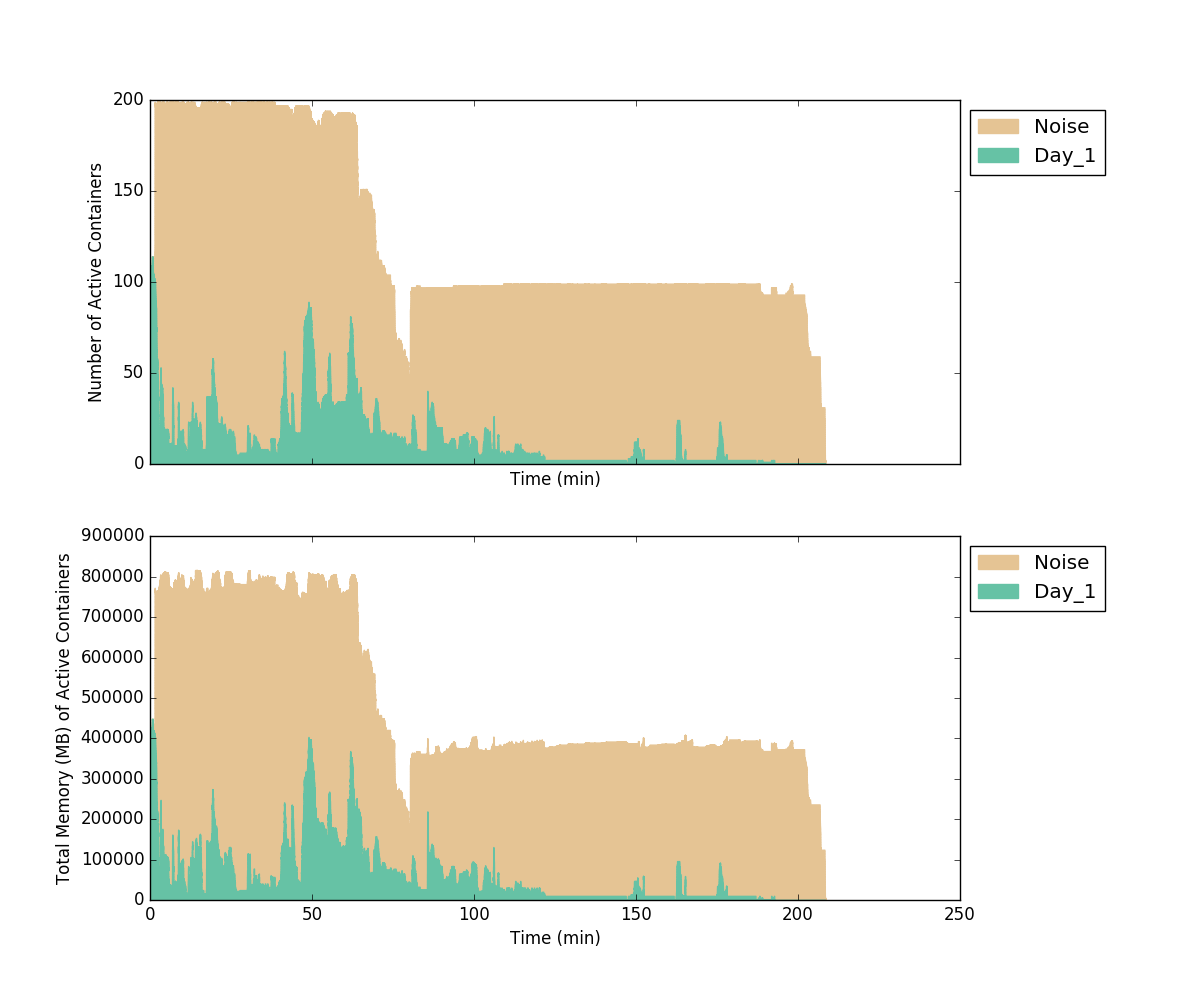
#### 5.3.3.1 Single SLA without Planner with reduced capacity Graph

As you can see, without planner, under this scenario, TCC hi\_cloud workflow continues running after more than 6 hours, beyond original deadline.



#### 5.2.3.2 Single SLA with Planner and reduced capacity Graph

With the planner, even under the same heavy non-critical workloads, we were able to complete them around deadline with reduced cluster capacity.



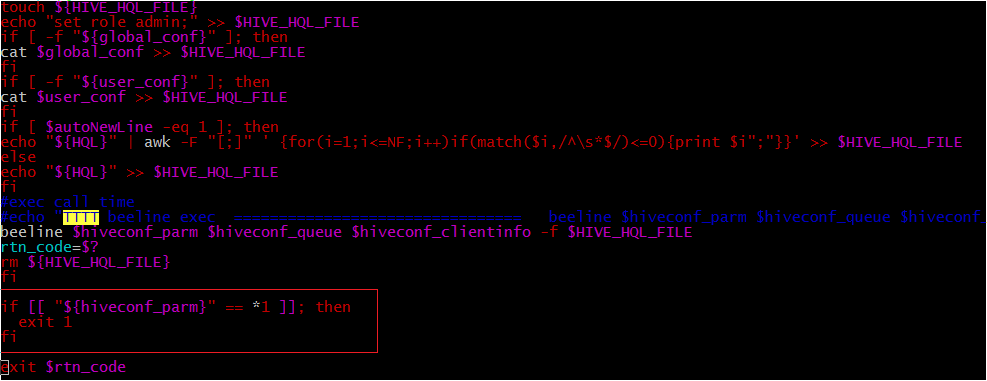
## 5.4 Case 4 Single SLA with application failure/retry

### 5.4.1 Case Description

TCC has built-in functionality to rerun failure tasks automatically. For instance, if Hive task fails, TCC will rerun failed tasks multiple times. Planner should be able to support task rerun behavior.

In order to simulate this case, we will run one day hi\_cloud workflow with noisy workloads together. We will fail some hive tasks. TCC should be able to rerun those failed tasks, Planner should follow on to still give guaranteed resources to those rerun workloads

### 5.4.2 Experiment Steps

1. Start with all systems turned off (adapter, tcc, planner, rm).
2. Modify hive script (/work/v/HIBI/hive). 

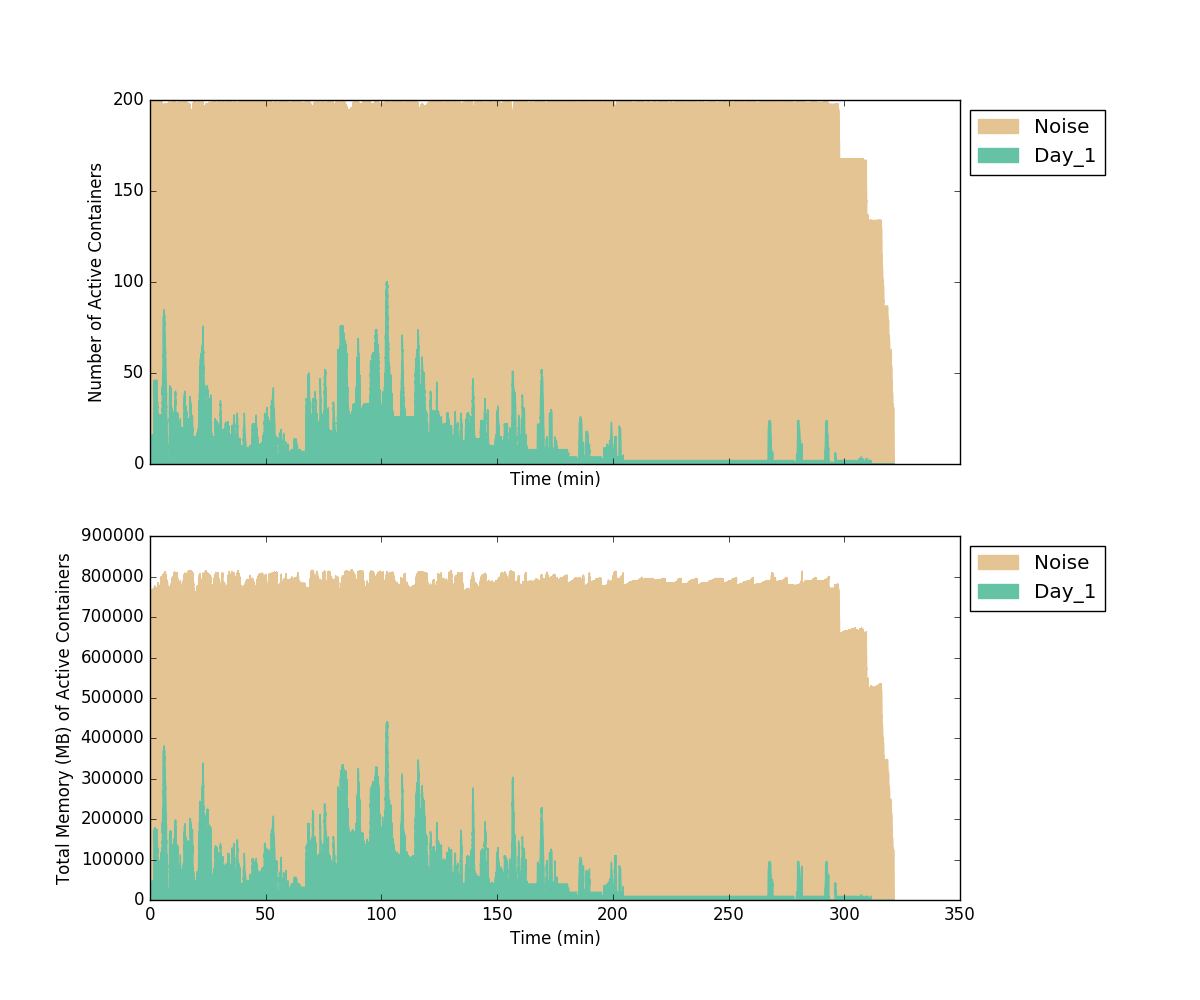
The above few lines were added for this case.

1. Start resource manager, check GUI to check that all 4 nodes have connected, that cluster has 200 vcores, 800GB memory, and that there are no jobs running. Each node should have 50 vcores, 200GB memory.
2. If running planner: turn on planner, check that it is communicating with the yarn cluster, and is reporting that the cluster has 200 vcores, 800GB memory.
3. Set all tasks in TCC database to done.
4. Set day 1 (in this case, it was 20170201) in TCC database to initialized.
5. Turn on noise maker script, with 15 instances, 0 delay between runs. Check that the cluster is completely saturated before continuing.
6. If running planner: turn on TCC adapter, check that it reports that the workflow has 75 nodes, and the plan is submitted successfully.
7. Start TCC.
8. Wait for all jobs to finish.
9. Record run time from TCC GUI.
10. If desired: generate graphs to show cluster utilization.
11. Revert hive script
12. Turn off noise maker script. Run kill\_the\_noise.sh from another window, then either wait, or kill all of the noise applications in YARN.

### 5.4.3 Results (Passed)

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 4.1 | Single SLA with application failure/retry | hicloud\_day execution time: 5 hours 16 min 46 seconds |

This case shows that planner is able to recognize TCC task automatic rerun scenario and continues providing guaranteed resource to those high priority workflow until it completes.



## 5.5 Case 5 Single SLA with manual stopped TCC tasks

### 5.5.1 Case Description

If some workloads keep failing for whatever reasons, for instance data is corrupted, system administrators need to be involved to manually fix the problem. In such case, administrator may need to manually stop a few TCC tasks, then fix environment issues, later on resume them. Superior Planner should work with this scenario, basically when stopped TCC tasks are resumed, Superior Planner should recognize them and continue providing guarantee resources to those critical workflow.

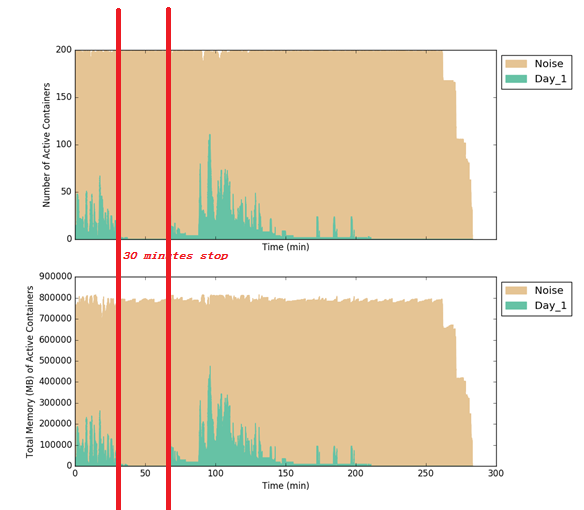
### 5.5.2 Experiment Steps

1. Start with all systems turned off (adapter, tcc, planner, rm).
2. Start resource manager, check GUI to check that all 4 nodes have connected, that cluster has 200 vcores, 800GB memory, and that there are no jobs running. Each node should have 50 vcores, 200GB memory.
3. If running planner: turn on planner, check that it is communicating with the yarn cluster, and is reporting that the cluster has 200 vcores, 800GB memory.
4. Set all tasks in TCC database to done.
5. Set day 1 (in this case, it was 20170201) in TCC database to initialized.
6. Turn on noise maker script, with 15 instances, 0 delay between runs. Check that the cluster is completely saturated before continuing.
7. If running planner: turn on TCC adapter, check that it reports that the workflow has 75 nodes, and the plan is submitted successfully.
8. Start TCC.
9. \*IMPORTANT\* Do NOT stop tasks 4092002 or 4040136 if they are currently running, as there is potential to cause these tasks to fail. After 30 minutes from the start time as reported by the TCC GUI, manually stop 4 tasks that are currently running by clicking “停止任务” in the TCC GUI. In this case, we stopped the running tasks 4093054, 4090021,4093059, and 4050034. After another 30 minutes, restart these tasks by clicking “启动任务” in the TCC GUI. Afterwards, make sure that those 4 tasks run correctly, and the mapreduce jobs run by those tasks are run in the correct queues.
10. Wait for all jobs to finish.
11. Record run time from TCC GUI.
12. If desired: generate graphs to show cluster utilization.
13. Turn off noise maker script. Run kill\_the\_noise.sh from another window, then either wait, or kill all of the noise applications in YARN.

### 5.5.3 Results (Passed)

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 5.1 | Single SLA with manual stopped TCC tasks | hicloud\_day execution time: 3 hours 33 minutes 47 seconds |

Since we stopped some critical tasks at 30 minutes after workflow starts to run, the stop lasted for 30 minutes duration, effectively, the entire workflow was postponed 30 minutes. With planner, it shows Planner can recognize this event and continue providing guaranteed resources and completes within 3 hours.



# 6. POC Conclusion

Consumer Business Group uses TCC system and FusionInsight product to manage and run their entire business workflow. Currently, they have challenge to ensure critical business workflows complete on time. IT CRC team research project (Superior SLA system) is designed to help solve this challenge. In order to verify the project effect, CBG and IT CRC team agreed to conduct a POC test based on subset of CBG workflows and dataset.

POC test cases are quite comprehensive and cover the most common scenarios, even including sudden cluster capacity reduction or job application failure cases. After the POC test, it is clearly concluded that Superior SLA system can greatly improve the overall system (TCC + FI cluster) to meet critical workflow deadlines. Superior SLA system has proven to be a very important and useful feature for the CBG use case. CBG suggests FI product team delivers the Superior SLA system in an official release.

POC 人员：

CBG : 王光明、张文虎、林啸鸣、罗向龙

IT 产品线：蔺若林、陈翀、陈祥

# Appendix

## A.1 Miscellaneous (Optional)

Most of the following settings were set while tuning the system to run TPC-H on Hive benchmark earlier in the year. I document them here to give complete picture of our env.

* hdfs-site.xml
  + dfs.namenode.handler.count = 40
  + dfs.namenode.safemode.threshold-pct = 1
  + dfs.datanode.du.reserved = 1024000000
  + dfs.permission.enabled = false
* core-site.xml
  + io.file.buffer.size = 32768
* mapred-site.xml
  + mapreduce.task.io.sort.factor = 100
  + mapreduce.task.io.sort.mb = 200
  + mapreduce.reduce.shuffle.parallelcopies = 25
  + mapreduce.map.speculative = false
  + mapreduce.reduce.speculative = false
  + mapreduce.output.fileoutputformat.compress = false
  + mapreduce.map.output.compress = true
  + mapreduce.map.output.compress.codec = com.hadoop.compression.lzo.LzoCodec

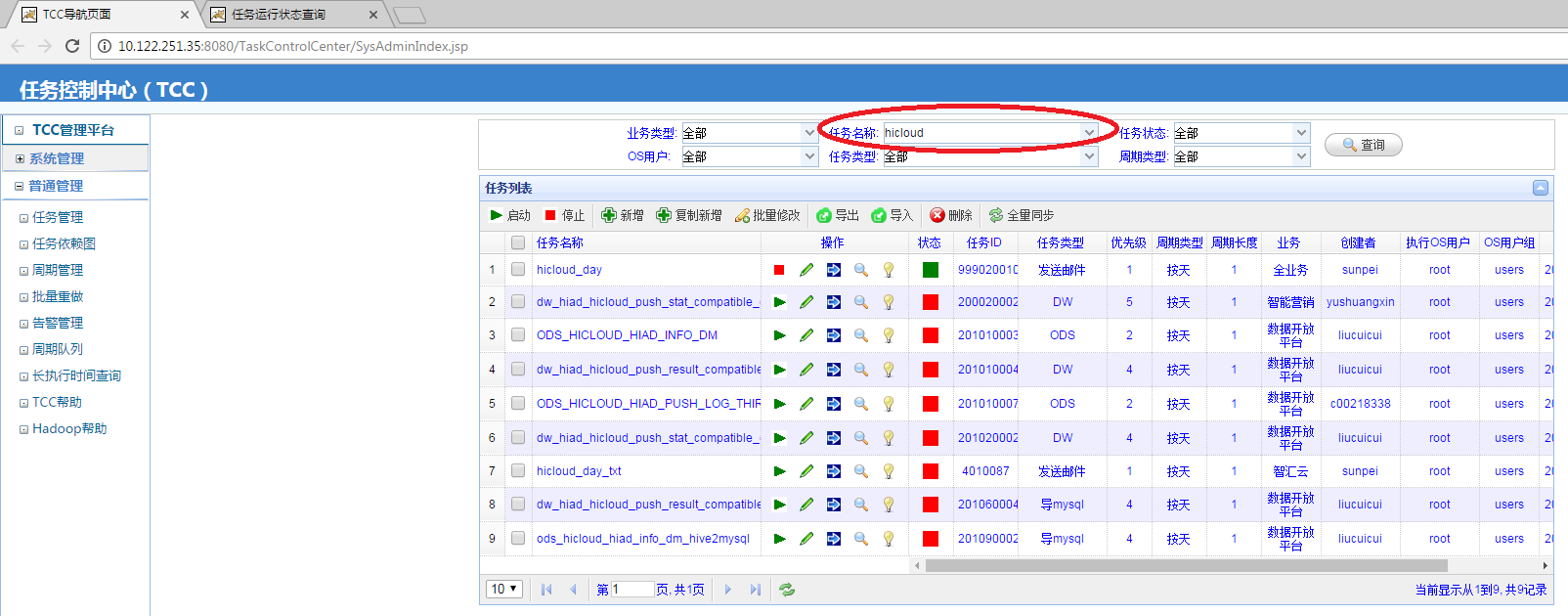
## A.2 Resetting System Components

To do a complete system reset, take the following steps:

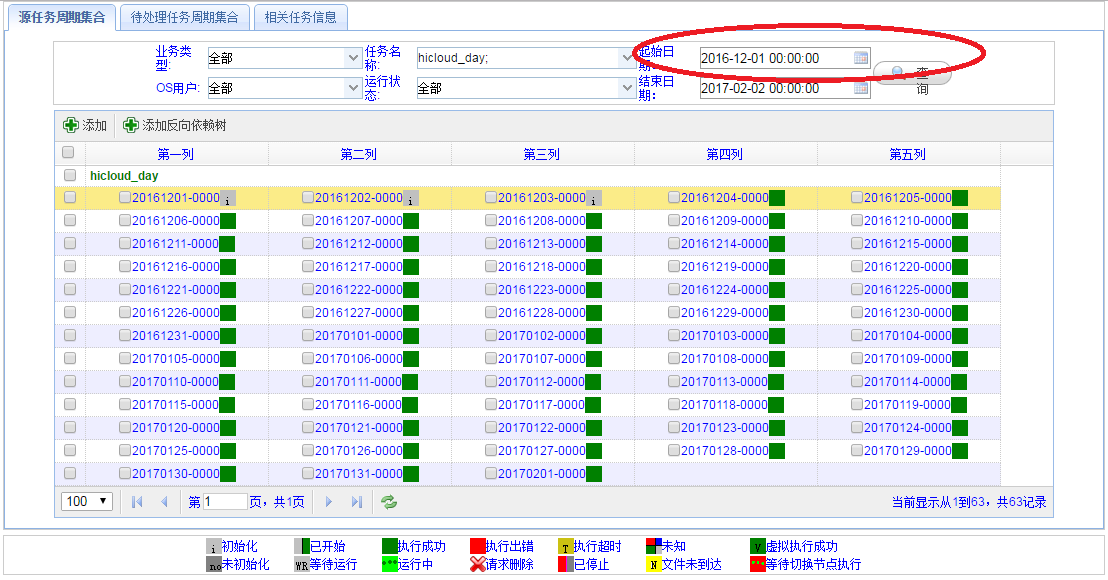
* Stop TCC
* Stop the TCC Adapter
* Stop the Planner
* Stop the ResourceManager
* Delete all contents of ${splanner.shared.dir}/predictor
* Start the ResourceManager
* Start the Planner
* Reinitialize the TCC Database tasks as above
* Start the TCC Adapter
* Start TCC

## A.3 How to check TCC workflow duration

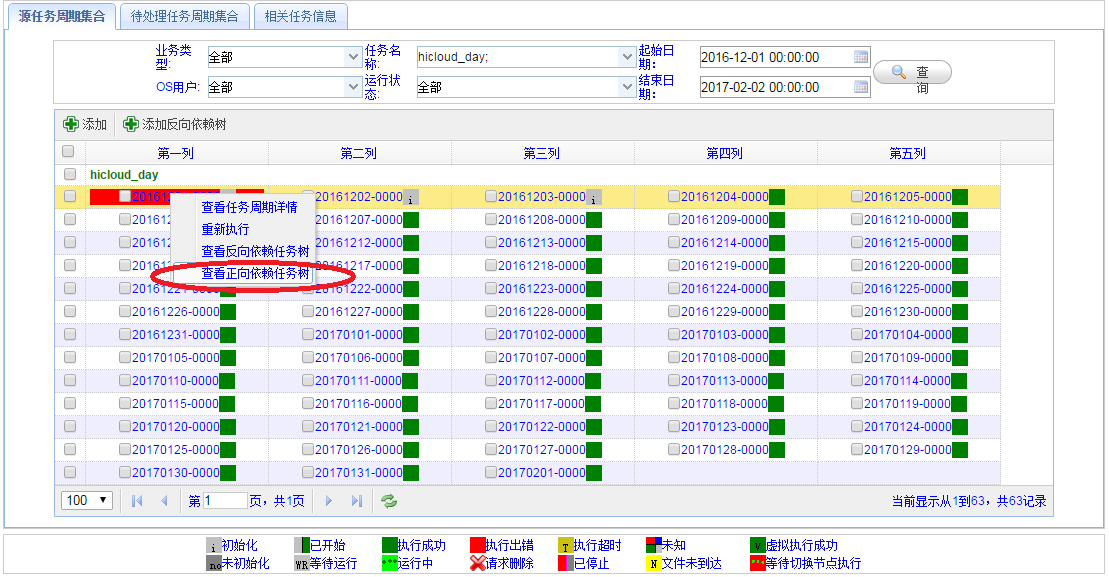
In order to check end to end workflow duration, one can login to TCC, in task control center page, input “hicloud” to search related TCC workflow as below:



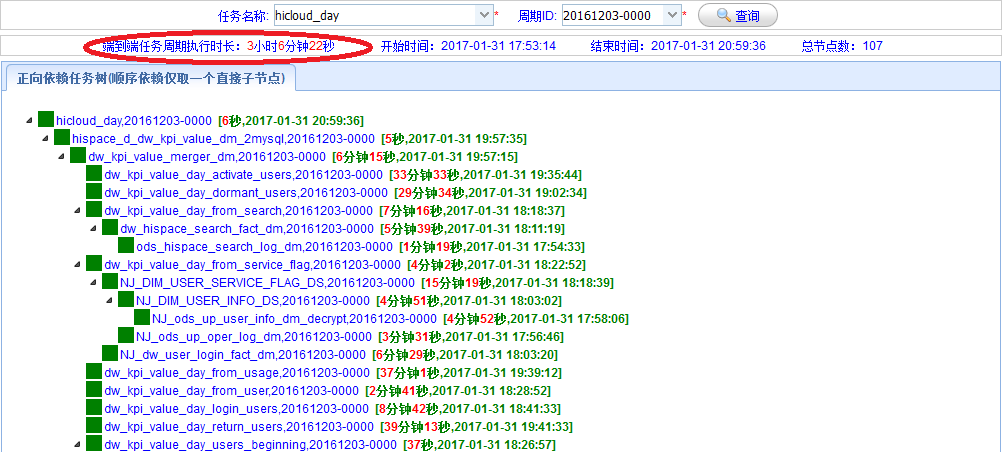
In “Task running status” page, select the right date time, which is “Dec 1st, 2016” in our case:



Then select the date, right click mouse to select “查看正向依赖任务树”:



Then you will see this instance of workflow and its total end to end execution time. See red circle, in this case it is “3 hours, 6 minutes, 22 seconds”.



## A.4 How to run graph generation tool to show resource utilization result

We ran:

$SPARK\_HOME/bin/spark-submit --class "com.huawei.superior.planner.prediction.ParseMRTasksOverTimeClient" --master yarn lib/superior-planner-predict-engine.jar "hdfs:///tmp/hadoop-yarn/staging/history/done/2017/02/02/000000/job\_{RM\_TIMESTAMP}\*conf.xml" "hdfs:///tmp/hadoop-yarn/staging/history/done/2017/02/02/000000/job\_{RM\_TIMESTAMP}\*.jhist" file:/// path/to/results/batchname

Replace {RM\_TIMESTAMP} with the timestamp of your rm. Restarting your rm after each run will give each run its own unique rm timestamp, and makes graph generation easier.

Then run:

./plot\_active\_containers.workflow.py /path/to/results/batchname --action\_table action\_table.txt –m –n

Add the –n when running with noise, otherwise it is not necessary.

A full guide can be found in the README under superior-planner-predict-engine/src/test/resources/VisualizationTools

# A.5 Canada Env Test Results

Cluster Info

DataSight Hadoop V100R002C10SPC200B103

Total 320 vcores, 1.28TB Cluster resource, 4 Worker nodes

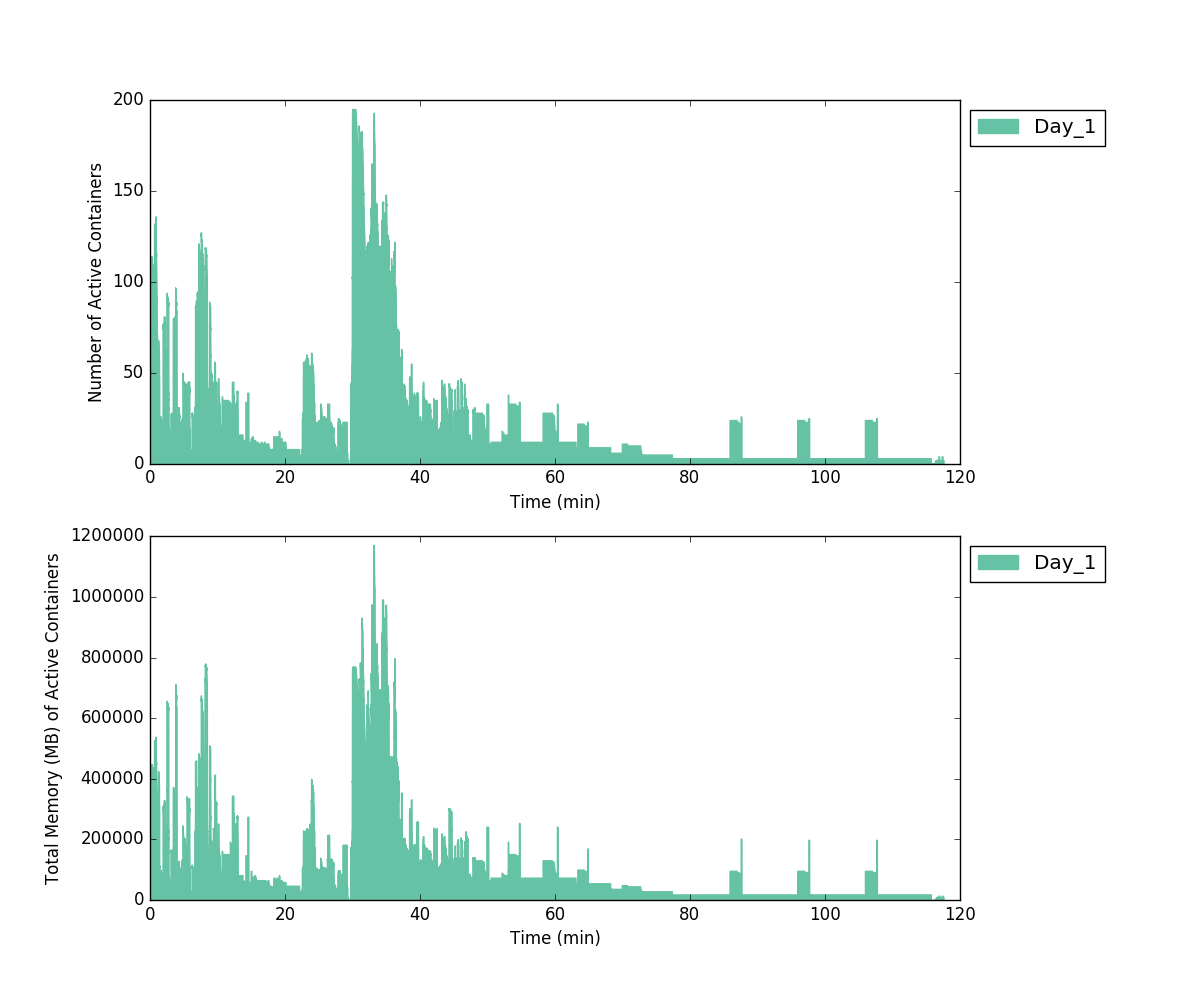
DC3.0 TCC HiCloud Workflow

38GB Business raw data

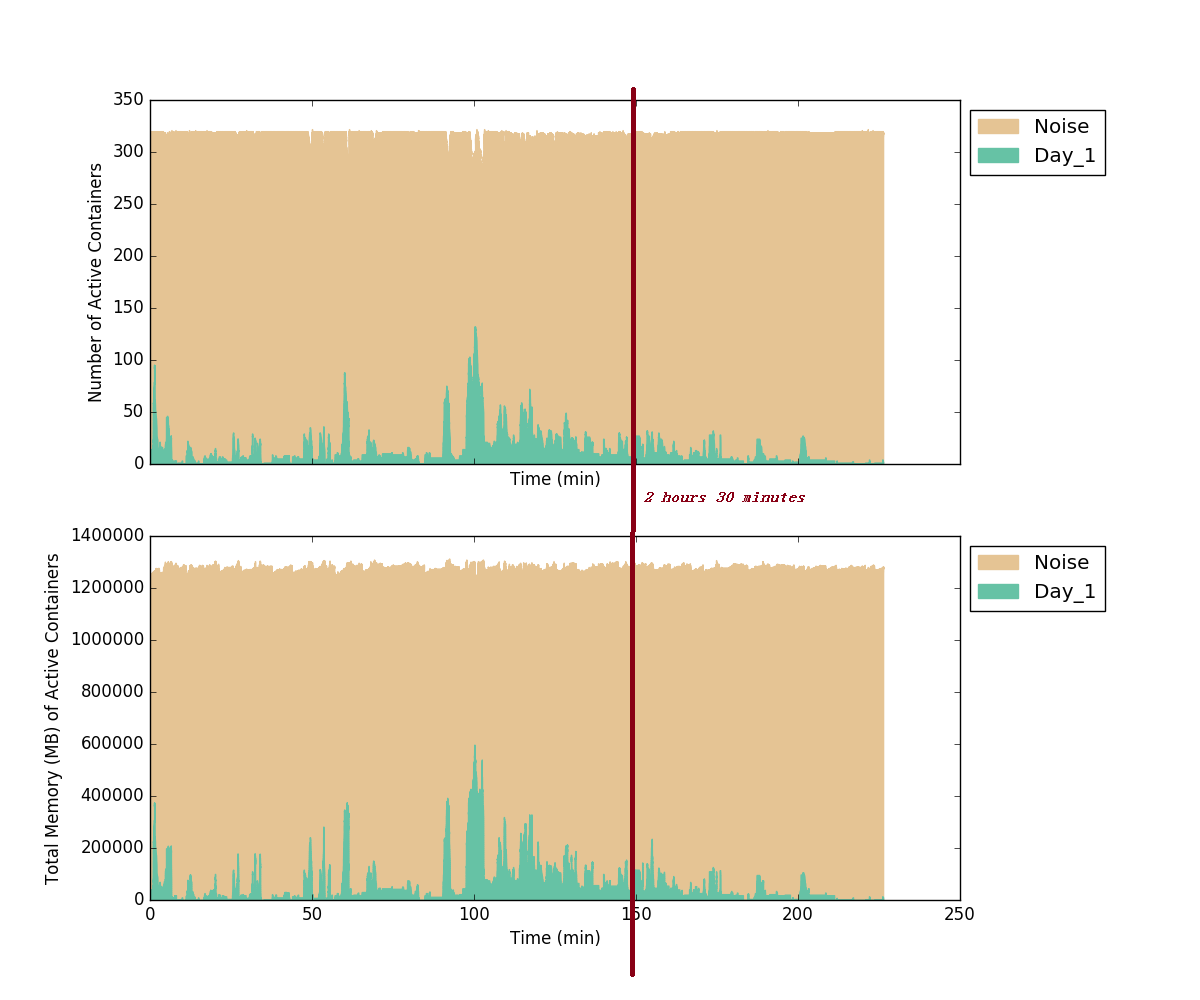
## A.5.1 Case 1 Single workflow with SLA deadline

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 1.1 | Single SLA: No Planner, Quiet | hicloud\_day execution time: 1h 58m 31s |
| 1.2 | Single SLA: No Planner, Noisy | hicloud\_day execution time: 3h 45m 33s |
| 1.3 | Single SLA: Planner with Noisy | hicloud\_day execution time: 2h 20m 02s |

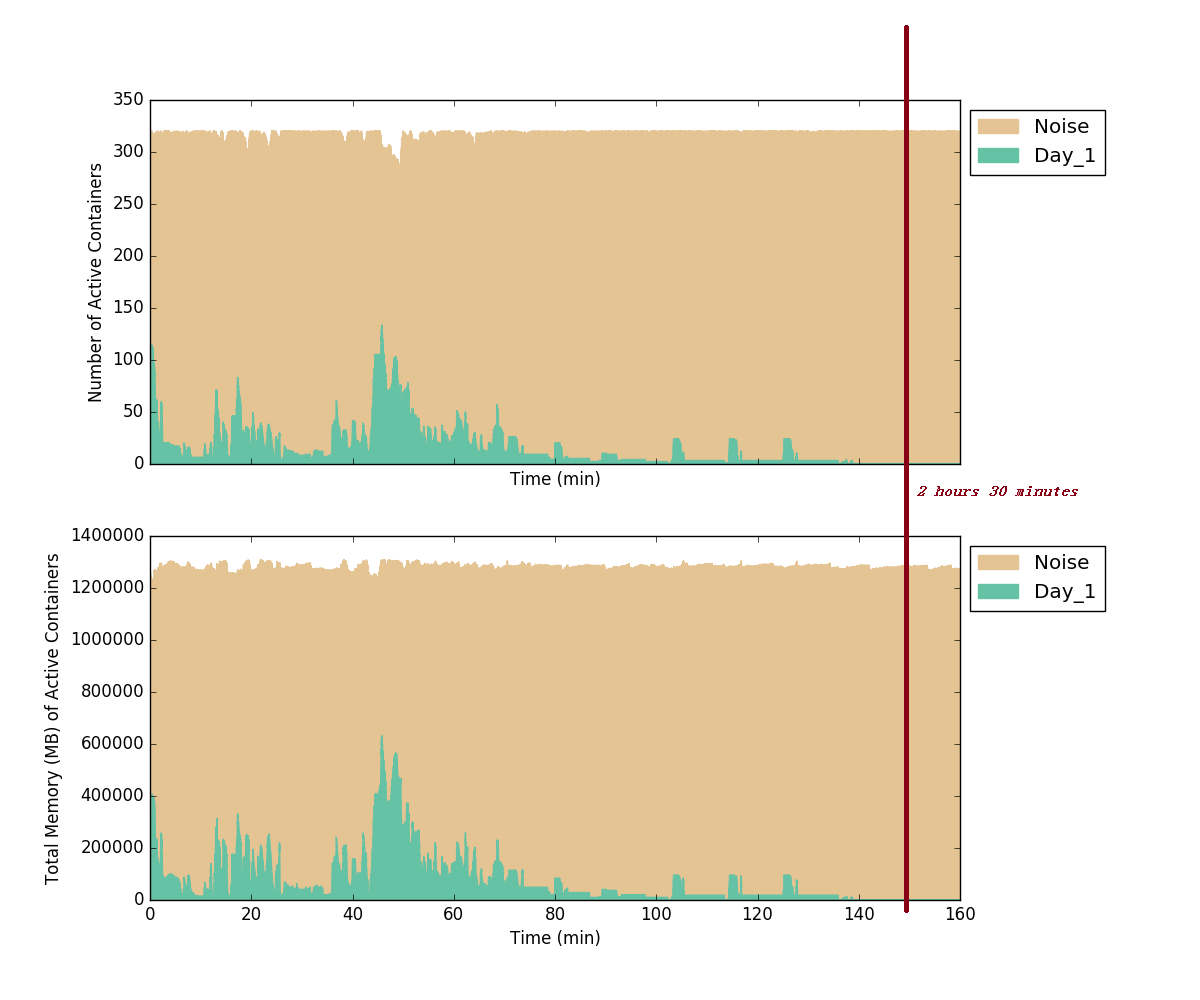
### A.5.1.1 Single SLA: No Planner, Quiet Graph



### A.5.1.2 Single SLA: No Planner, Noisy Graph



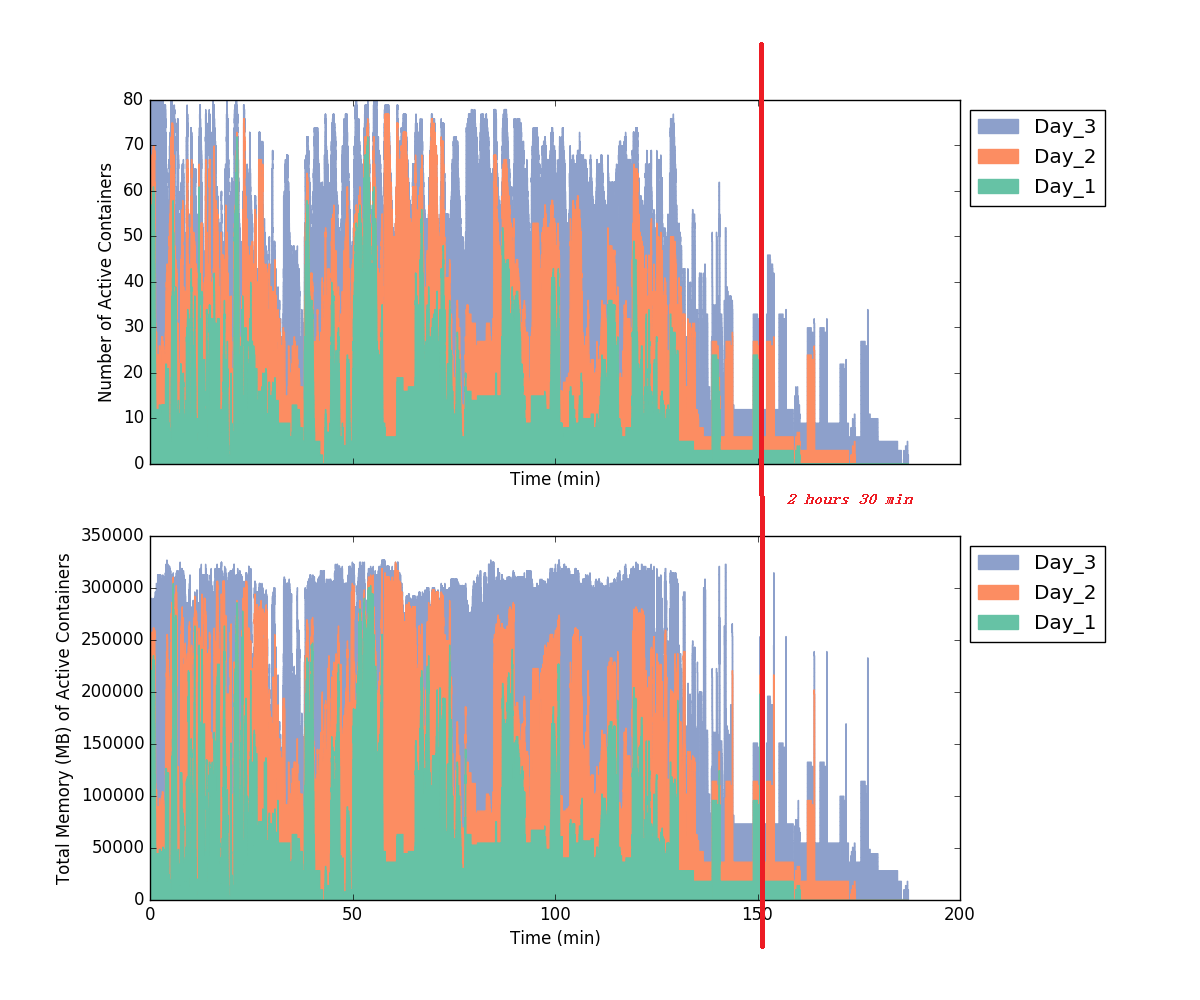
### A.5.1.3 Single SLA: Planner, Noisy Graph



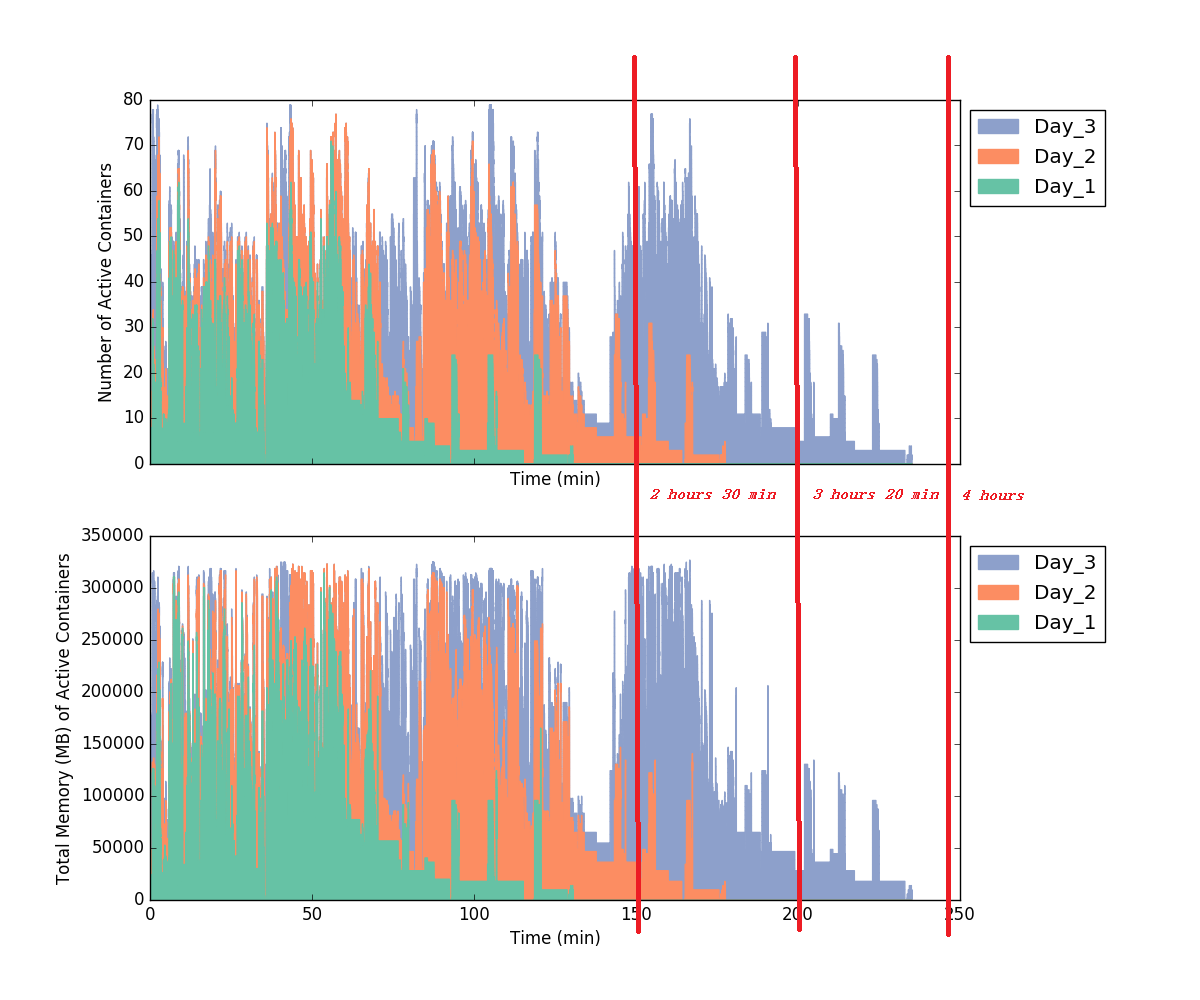
## A.5.2 Case 2 Multiple workflows with different SLA deadline

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 2.1 | Multiple SLA no Planner | **hicloud\_day 1: 2 hours 41 min 22 seconds**  hicloud\_day 2: 2 hours 55 min 02 seconds  hicloud\_day 3: 3 hours 8 min 12 seconds |
| 2.2 | Multiple SLA with Planner | hicloud\_day 1: 2 hours 11 min 22 seconds  hicloud\_day 2: 2 hours 58 min 22 seconds  hicloud\_day 3: 3 hours 56 min 02 seconds |

### A.5.2.1 Multiple SLAs without Planner Graph



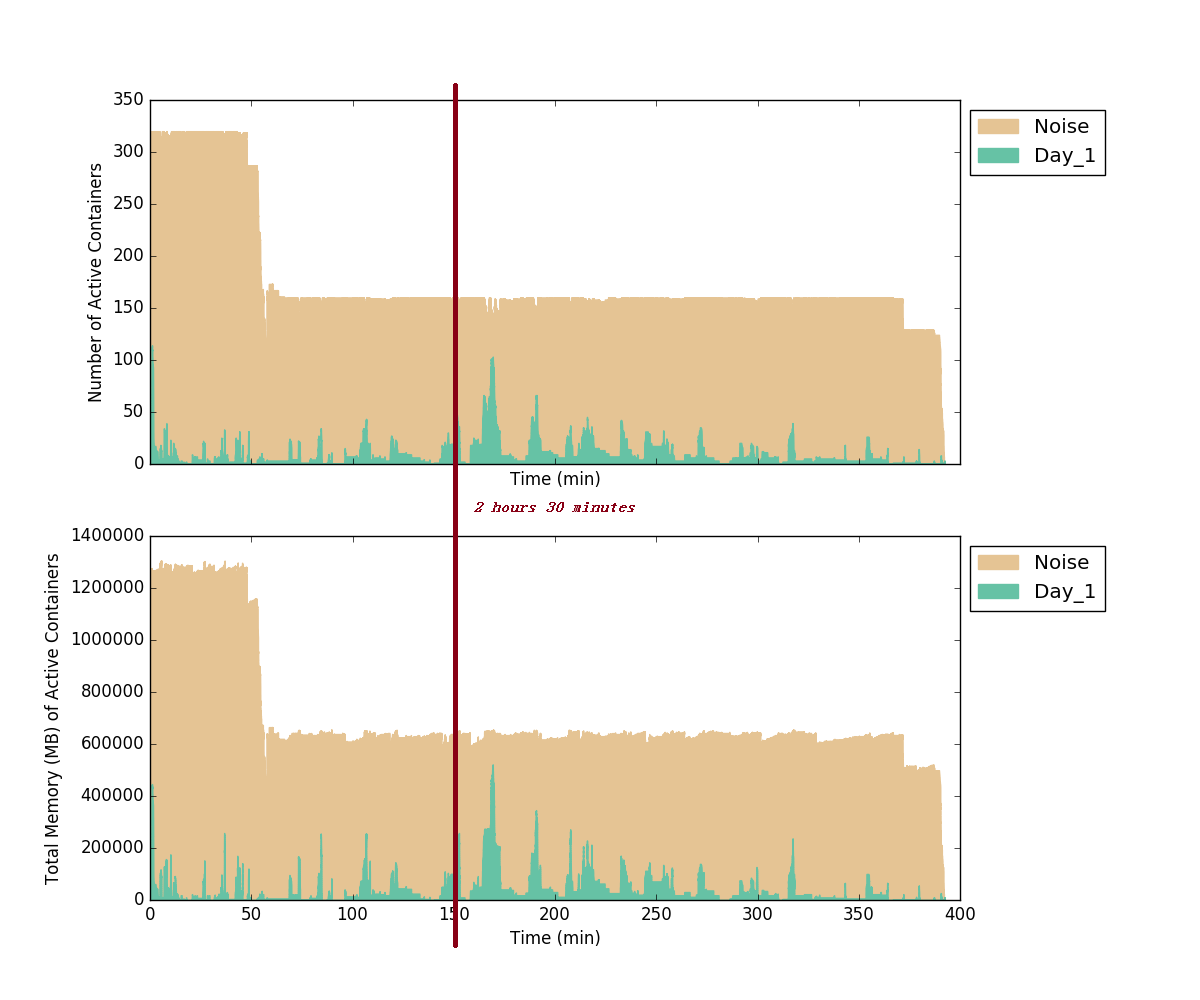
### A.5.2.2 Multiple SLAs with Planner Graph



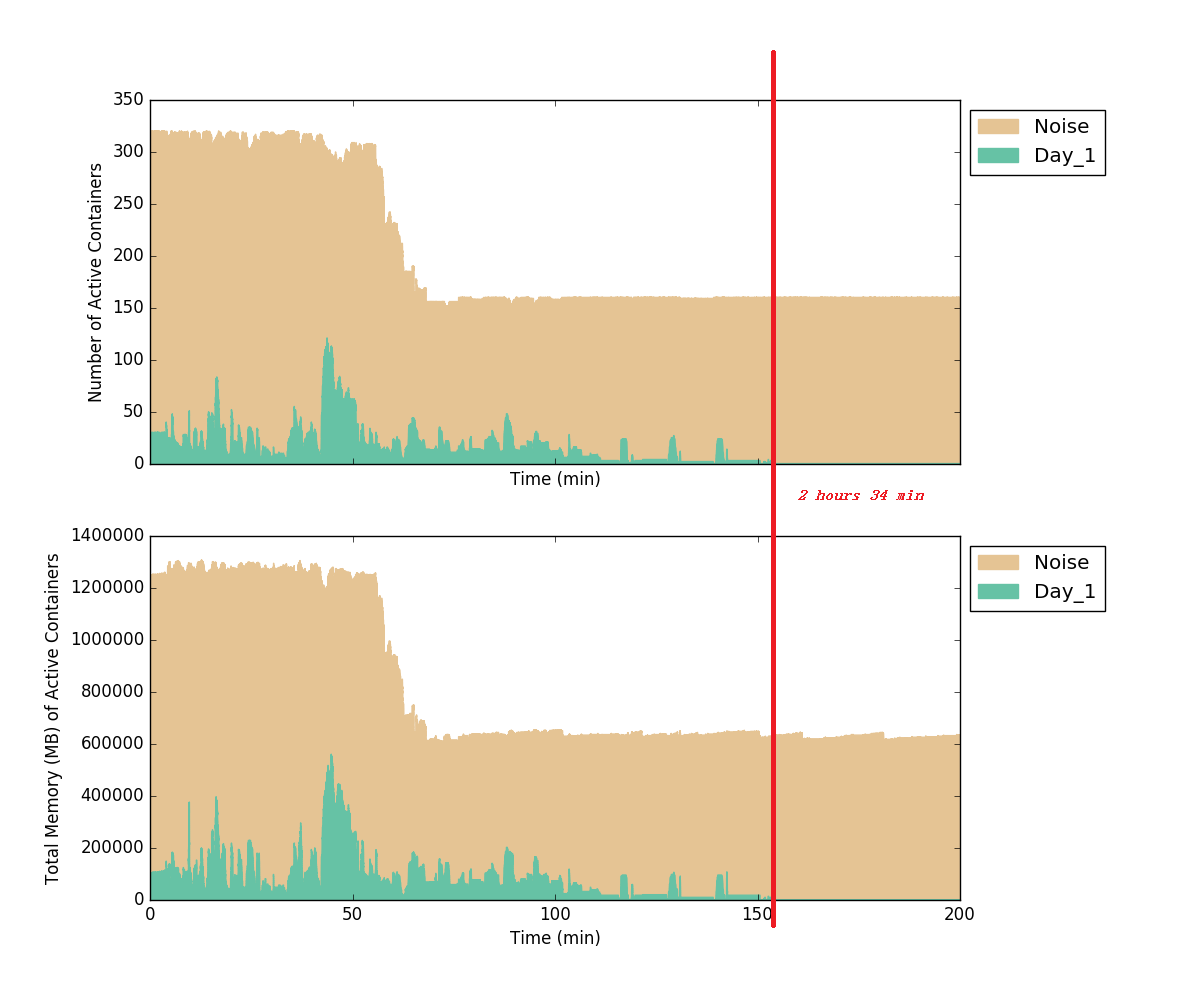
## A.5.3 Case 3 Single SLA with reduced Capacity

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 3.1 | Single SLA no Planner with reduced capacity | hicloud\_day execution time: more than 6 hours, still running |
| 3.2 | Single SLA with Planner and reduced capacity | hicloud\_day execution time: 2 hours 34 min 53 seconds |

### A.5.3.1 Single SLA without Planner with reduced capacity Graph

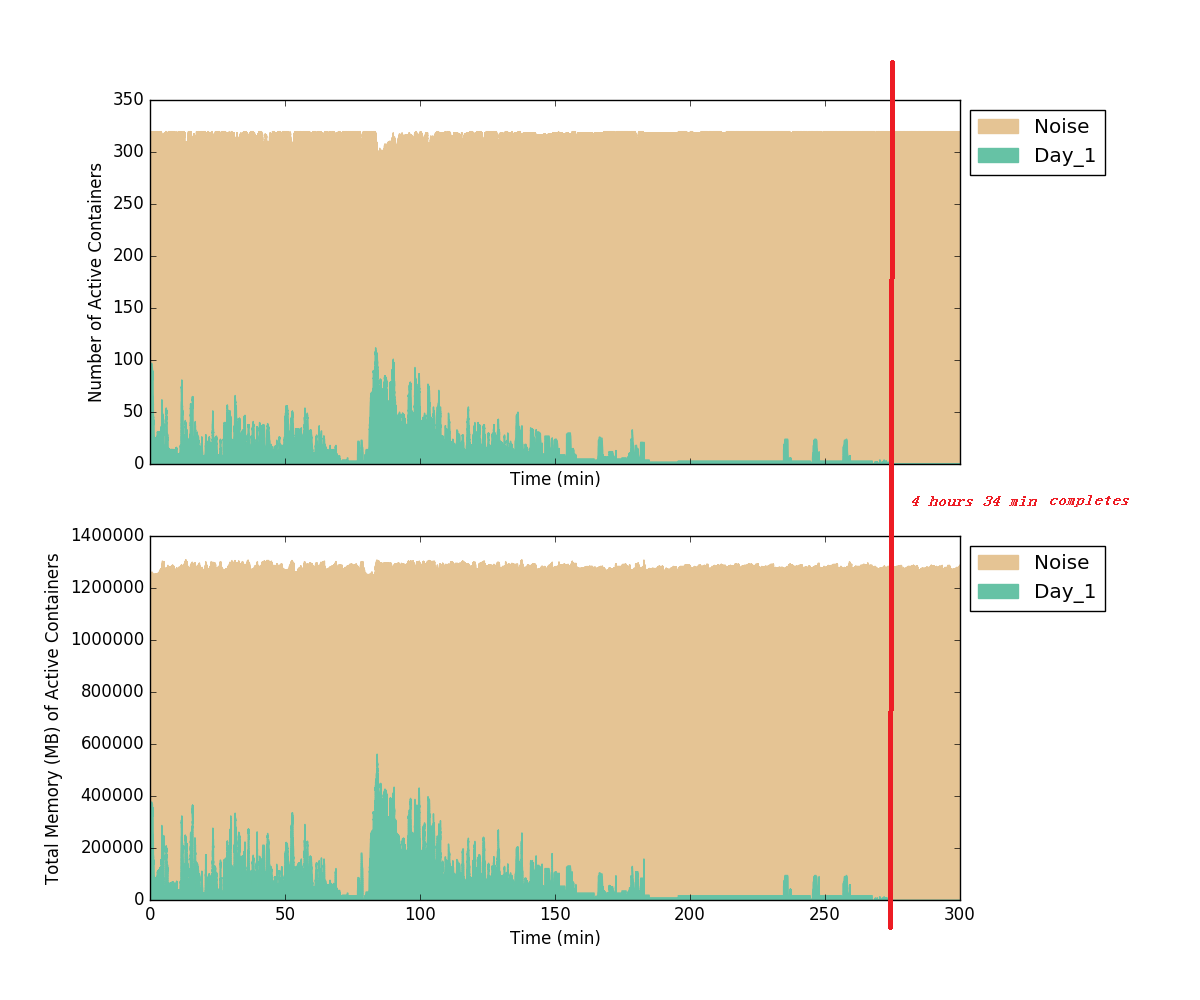


### A.5.3.2 Single SLA with Planner and reduced capacity Graph



## A.5.4 Case 4 Single SLA with application failure/retry

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 4.1 | Single SLA with application failure/retry | hicloud\_day execution time: 4 hours 34 min 29 seconds |



## A.5.5 Case 5 Single SLA with manual stopped TCC tasks

|  |  |  |
| --- | --- | --- |
| # | Case Name | Result Summary |
| 5.1 | Single SLA with manual stopped TCC tasks | hicloud\_day execution time: 2 hours 59 minutes 3 seconds |

