patch_analysis

January 6, 2022

[1]:
```
from pymongo import MongoClient
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from more_itertools import pairwise
import requests
import json
import yaml
import os
import datetime
from jupyter_datatables import init_datatables_mode

user=os.getenv("PERF_DB_READ_USER")
password=os.getenv("PERF_DB_READ_PASSWORD")

# total number of tasks to read from REST (about 20k in all)
max_tasks = 20000

# total number of tests to process for mean & std. dev.
max_tests = 20000

# of tasks per REST call
batch = 100

#Drew's v4 impact Patch
#build_a = '61b2af58e3c33129ecd80fb4'
#build_b = 'sys_perf_1730e8918556524e0207c00869328df7d40f4ff5'

#louis "don't ruin perf patch"
#build_a = '61ae2c9b3e8e862ec34fd63b'
#build_b = 'sys_perf_77c9521d20f7a6dd0f3d115bf4f619968e440830'

#Greg's
build_a = '61d4b17730661507f680da18'
build_b = 'sys_perf_91d9b5a0d92784178c1f01e6a5766e6366f1c797'

#Dan Gomezferro
```

```
#build_a = '61d2c806850e61439388c2e2'
#build_b = 'sys_perf_0b5f8fbf748fa7c8da75bd64ac9ca4ed322de321'
#stable point sanity check
#build_a = 'sys_perf_43479818bd01f27ee25b6e992045529d2ac0185a'
#build_b = 'sys_perf_43479818bd01f27ee25b6e992045529d2ac0185a'
build_a_label = 'Patch'
build_b_label = 'Base Commit'

# connection strings
evg_base = "https://cedar.mongodb.com/rest/v1/perf/version"
client = MongoClient(f"mongodb+srv://{user}:{password}@performancedata-g6tsc.
 ↪mongodb.net/expanded_metrics?
 ↪readPreference=secondary&readPreferenceTags=nodeType:
 ↪ANALYTICS&readConcernLevel=local")

date_b = client["expanded_metrics"]["versions"].find_one({"version_id":␣
 ↪build_b})["commit_date"]
date_a = date_b
```

[2]:
```
%%time

# Read the list of tasks for the 2 commits from REST

print("Reading a ", build_a_label)

skip = 0
data = {
    "project": [],
    "variant": [],
    "task": [],
    "test": [],
    "measurement": [],
    "args": [],
    "execution": [],
    "value": [],
    "task_id": [],
}

while True:
    r = requests.get(f"{evg_base}/{build_a}?skip={skip}&limit={batch}")
    if r.status_code == 404:
        break

    for cp in r.json():
        if cp["rollups"]["stats"]:
            for rollup in cp["rollups"]["stats"]:
```

```
                data["project"].append(cp["info"]["project"])
                data["variant"].append(cp["info"]["variant"])
                data["task"].append(cp["info"]["task_name"])
                data["test"].append(cp["info"]["test_name"])
                data["measurement"].append(rollup["name"])
                data["args"].append(cp["info"]["args"])
                data["execution"].append(cp["info"]["execution"])
                data["value"].append(rollup["val"])
                data["task_id"].append(cp["info"]["task_id"])
    print(skip, end='\r')
    skip += batch
    if skip > max_tasks:
        break

dfa = pd.DataFrame(data=data)
```

```
Reading a  Patch
CPU times: user 599 ms, sys: 50.2 ms, total: 649 ms
Wall time: 1.39 s
```

[3]:
```
%%time

# Filter and merge the tasks from the 2 commits

def filter_canaries(dframe):

    # ^((?!
 ↪canary_|fio_|iperf|NetworkBandwidth|[01]_1c_avg_latency|[01]_1c_max_latency|oplog1|finishin
    #␣
 ↪Setup|Quiesce|GennyOverhead|ShardCollection|EnableSharding|genny_canaries|nop_).
 ↪*(?<!Setup|ActorFinished|
    # ActorStarted))$

    dframe_filtered = dframe[~dframe.test.str.
 ↪match('CleanUp|canary|fio|iperf|NetworkBandwidth|finishing|Setup|Quiesce|GennyOverhead')]
    dframe_filtered = dframe_filtered[~dframe_filtered.test.str.
 ↪contains('ActorFinished|ActorStarted|Setup')]
    return dframe_filtered

print('dfa length = ', len(dfa))

comparison = filter_canaries(dfa)
print('filtered ', len(comparison))

comparison["args"]= comparison["args"].apply(json.dumps)
```

```
found_ts = comparison[["project","variant","task","test","measurement","args",␣
 ↪"task_id"]]

# We drop duplicates since there could be multiple executions for the same␣
 ↪combination of the properties below.
found_ts = found_ts.drop_duplicates()

print('length after de-dup = ', len(found_ts))

# keep the interesting metrics
found_ts = found_ts[found_ts["measurement"].isin(['AverageLatency'
                                                  ,'ops_per_sec'
                                                 #  ,'system cpu user (%) - mean'
                                                 #  ,'ss mem resident (MiB) -␣
 ↪mean'
                                                 #  ,'Data - disk xvde␣
 ↪utilization (%) - mean'
                                                 #  ,'Journal - disk xvdf␣
 ↪utilization (%) - mean'
                                                  ])]

print('length after keeping interesting metrics = ', len(found_ts))
```

```
dfa length =  15599
filtered  13277
length after de-dup =  13277
length after keeping interesting metrics =  710
CPU times: user 86.3 ms, sys: 8.75 ms, total: 95 ms
Wall time: 109 ms
```

[4]:
```
# From Alex Costas: Algorithm to look up time series from the anaytics node in␣
 ↪able to characterize
# the stable region of results around build_a.

def get_stable_region(commit_date, ts, cps):
    true_positive_orders = {
        cp["order"]
        for cp in cps
        if cp["triage"]["triage_status"] == "true_positive"
    }
    len_ts = len(ts["data"])
    stable_region_bounds = (
        [0]
        + [idx for idx, datum in enumerate(ts["data"]) if datum["order"] in␣
 ↪true_positive_orders]
        + [len_ts]
    )
```

```
    start = end = 0

    # if base commit before or after the entire time series, get the closest␣
↪stable region
    if commit_date < ts["data"][0]["commit_date"]:
        # first stable region
        start = stable_region_bounds[0]
        end = stable_region_bounds[1]

    if commit_date > ts["data"][len_ts - 1]["commit_date"]:
        # last stable region
        start = stable_region_bounds[-2]
        end = stable_region_bounds[-1]

    for start_bound, end_bound in pairwise(stable_region_bounds):
        if (
            ts["data"][start_bound]["commit_date"]
            <= commit_date
            <= ts["data"][end_bound-1]["commit_date"]
        ):
            start = start_bound
            end = end_bound
            # find center
            center = start
            for datum in ts["data"][start:end]:
                if datum["commit_date"] <= commit_date:
                    center = center + 1
            start=max(start, center-10)
            end=min(end, center+10)
    return [datum["value"] for datum in ts["data"][start:end]]
```

[5]:
```
%%time



# Calculate the means and std dev for the Zscores
# Must be on VPN to read the analytics DB
print('')
# limit number of tests
found_ts = found_ts[0:max_tests]

total = len(found_ts)

stable_mean = []
stable_std = []
stable_length = []
```

```
for index, row in found_ts.iterrows():
    # some tests do not have threads.
    if row["args"] == "null":
            row["args"] = "{}"
    ts = client["expanded_metrics"]["time_series"].find_one({
            "project": row["project"],
            "variant": row["variant"],
            "task": row["task"],
            "test": row["test"],
            "args": json.loads(row["args"]),
            "measurement": row["measurement"],
        })
    cps = list(client["expanded_metrics"]["change_points"].find({
            "time_series_info.project": row["project"],
            "time_series_info.variant": row["variant"],
            "time_series_info.task": row["task"],
            "time_series_info.test": row["test"],
            "time_series_info.args": json.loads(row["args"]),
            "time_series_info.measurement": row["measurement"],
    }))

    try:
      stable_region = get_stable_region(date_a, ts, cps)
      stable_mean.append(np.mean(stable_region))
      stable_std.append(np.std(stable_region))
      stable_length.append(len(stable_region))
    except:
        # no stable region found
        print('')
        print('no stable region found for ', len(stable_length))
        print('')
        stable_mean.append(np.nan)
        stable_std.append(np.nan)
        stable_length.append(0)
        pass

    print('{}/{}'.format(len(stable_length), total), end='\r')

print('')

found_ts.insert(0, "stable_mean", stable_mean)
found_ts.insert(1, "stable_std", stable_std)
found_ts.insert(2, "stable_length", stable_length)


# drop charts where stable length < 4
```

```
found_ts = found_ts[found_ts["stable_length"] > 3]

print('after dropping stable_length < 4', len(found_ts))
```

```
710/710
after dropping stable_length < 4 702
CPU times: user 1.43 s, sys: 155 ms, total: 1.58 s
Wall time: 3.75 s
```

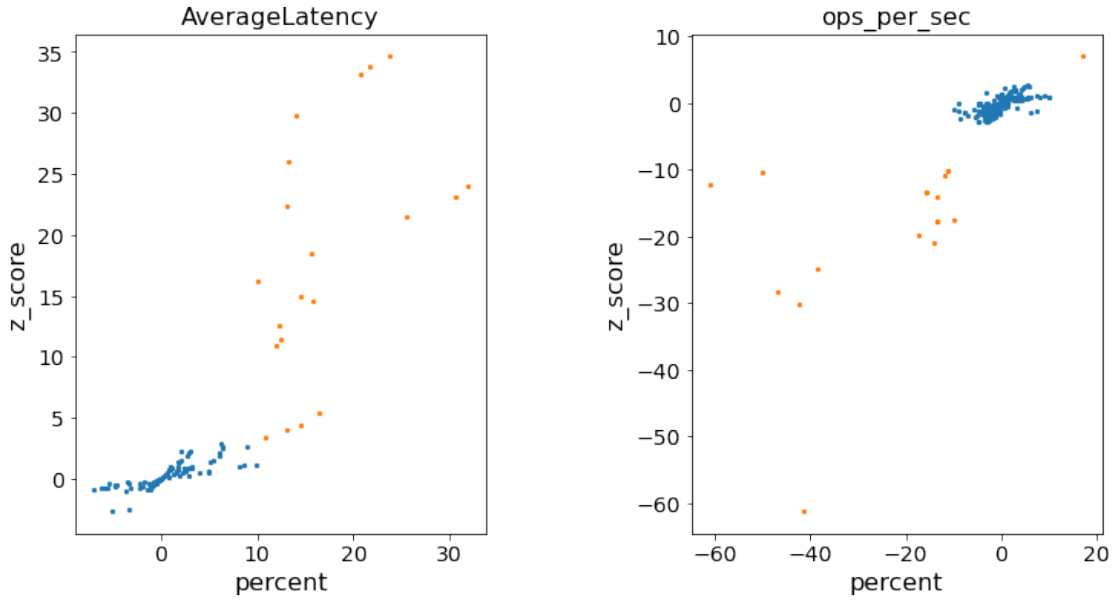
[6]:
```
# merge the results together:
comparison = comparison.merge(found_ts,␣
 ↪on=["project","variant","task","test","measurement","args", "task_id"])
comparison["percent"] = ((comparison["value"] / (1.
 ↪E-3+comparison["stable_mean"])) * 100) - 100
comparison["z_score"] = (comparison["value"] - comparison["stable_mean"]) / (1.
 ↪E-3+comparison["stable_std"])
```

[7]:
```
# save the data to CSV
with open("compare.csv", "w") as csv:
    comparison.to_csv(csv)
```

[8]:
```
# start from here to read cached data in compare.csv
# will also skip all the DB reads on the VPN

# need to run #1 to define imports and labels
# need to be on the VPN to get all the points for the charts.

with open("compare.csv", "r") as csv:
    comparison = pd.read_csv(csv)
```

[9]:
```
# scatter plots

# %matplotlib widget
# %matplotlib ipympl

# %matplotlib inline
# loses Engineering format
# import mpld3
# mpld3.enable_notebook()
from matplotlib.ticker import EngFormatter

params = {'legend.fontsize': 'large',
          'figure.figsize': (12, 12),
         'axes.labelsize': 16,
         'axes.titlesize': 16,
```

```
         'xtick.labelsize':14,
         'ytick.labelsize':14
         }
plt.rcParams.update(params)

fig, axs = plt.subplots(1,2, figsize=(12,6))
fig.subplots_adjust(hspace = .5, wspace=.5)

axs = axs.ravel()
i=0
for t in ['AverageLatency'
,'ops_per_sec'
#,'system cpu user (%) - mean'
#,'ss mem resident (MiB) - mean'
#,'Data - disk xvde utilization (%) - mean'
#,'Journal - disk xvdf utilization (%) - mean'
         ]:
    axs[i].yaxis.set_major_formatter(EngFormatter())
    axs[i].set_title(t)
    axs[i].set(xlabel="percent", ylabel="z_score")
    safe = comparison.query("(percent < 10 & z_score < 3) & (percent > -10 &␣
 ↪z_score > -3)")
    unsafe = comparison.query("(percent > 10 & z_score > 3) | (percent < -10 &␣
 ↪z_score < -3)")

    axs[i].scatter(safe["percent"][(safe["measurement"] == t)],
             safe["z_score"][(safe["measurement"] == t)], s=5)
    axs[i].scatter(unsafe["percent"][(unsafe["measurement"] == t)],
             unsafe["z_score"][(unsafe["measurement"] == t)], s=5)
    i=i+1
```



[10]:
```
def plot_timeseries(row_num):
    from IPython import get_ipython

    project = unsafe.loc[row_num, 'project']
    variant = unsafe.loc[row_num, 'variant']
    task = unsafe.loc[row_num, 'task']
    test = unsafe.loc[row_num, 'test']
    measurement = unsafe.loc[row_num, 'measurement']
    args = unsafe.loc[row_num, 'args']
    value = unsafe.loc[row_num, 'value']
    z_score = unsafe.loc[row_num, 'z_score']
    percent = unsafe.loc[row_num, 'percent']
    stable_mean = unsafe.loc[row_num, 'stable_mean']
    stable_std = unsafe.loc[row_num, 'stable_std']

    time_series = client["expanded_metrics"]["time_series"].find_one(
        { "project": project,
          "variant": variant,
          "test": test,
          "task": task,
          "measurement": measurement,
         "args": json.loads(args)
        }
    )

    dates = [time_series_point["commit_date"] for time_series_point in␣
↪time_series["data"]]
```

```
    values = [time_series_point["value"] for time_series_point in␣
↪time_series["data"]]

    params = {'legend.fontsize': 'large',
          'figure.figsize': (14, 4),
         'axes.labelsize': 12,
         'axes.titlesize': 12,
         'xtick.labelsize':10,
         'ytick.labelsize':10}
    plt.rcParams.update(params)

    plt.suptitle(variant+' '+task+' '+test, fontsize=10)
    plt.title("z_score = {:.2f}".format(z_score)+"  percent = {:.2f}".
↪format(percent), fontsize=10, loc='left')
    plt.plot(dates, values)

    plt.xlabel("Commit Date")
    plt.ylabel(time_series["measurement"])

    # add marks for the commits
    plt.text(date_a, value, build_a_label, rotation=90, fontsize=10)
    plt.axvline(date_a, color="green", linestyle="dotted")
    plt.axhline(value, color="green", linestyle="dotted" )
    plt.axhline(stable_mean, color="purple", linestyle="dashdot" )
    plt.show()
```

[11]:
```
# increase size of output window
from IPython.core.display import display, HTML
display(height=800)

unsafe = comparison.query("(percent > 10 and z_score > 3) | (percent < -10 and␣
↪z_score < -3)")
unsafe = unsafe.sort_values(by=['z_score', 'percent'], ignore_index=True)


df = pd.DataFrame(unsafe)
pd.set_option('display.max_rows', None)
pd.set_option('max_colwidth', 800)
pd.options.display.float_format = '{:.2f}'.format
# qgrid floating format
pd.set_option('display.precision', 3)


ddf = df[[ 'variant', 'task', 'test', 'measurement',  'z_score', 'percent',
          'value', 'stable_mean', 'stable_length','args', 'task_id']]
```

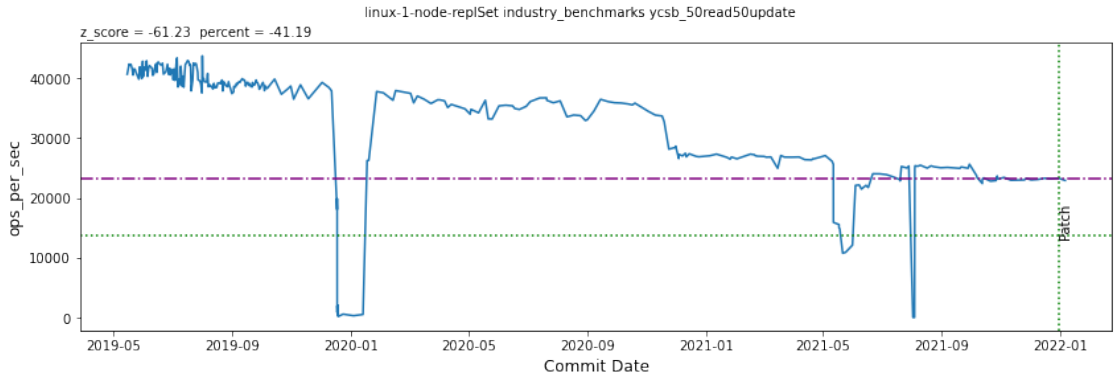
```
# save to disk
with open(f"selected_tasks_{build_a_label}_{build_b_label}.csv", "w") as csv:
    ddf.to_csv(csv)


import ipydatagrid



info_grid = ipydatagrid.DataGrid(ddf,base_row_size=32, base_column_size=100,␣
 ↪selection_mode="cell")
info_grid
```

```
DataGrid(auto_fit_params={'area': 'all', 'padding': 30, 'numCols': None},␣
 ↪base_column_size=100, base_row_size=…
```

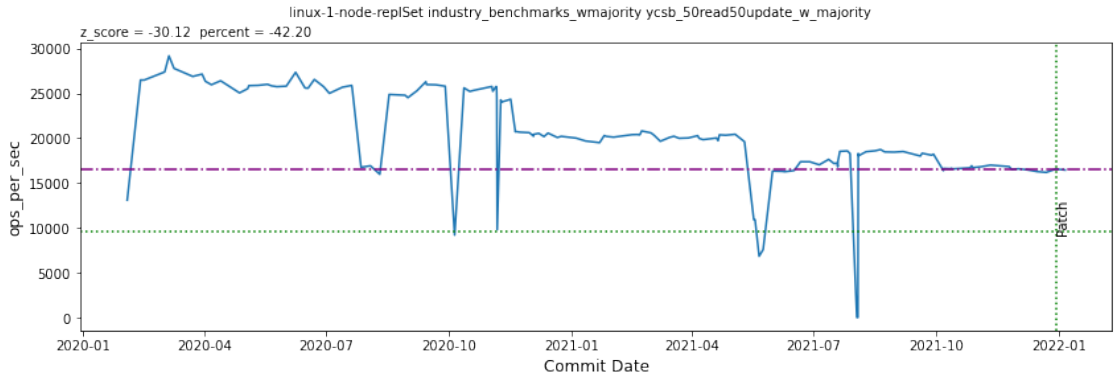
[ ]:

[12]:
```
class RenderHyperlink(object):
    def __init__(self, key, link, *args):
        link = link + "/" if not link.endswith("/") else link

        for arg in args:
            link += arg + "/"

        self.__url = "<a href={}>{}</a>".format(link, key)

    def __repr__(self):
        from IPython.core.display import display, HTML
        display(HTML(self.__url))
        return "" # hacky way to return a string despite not returning anything
```

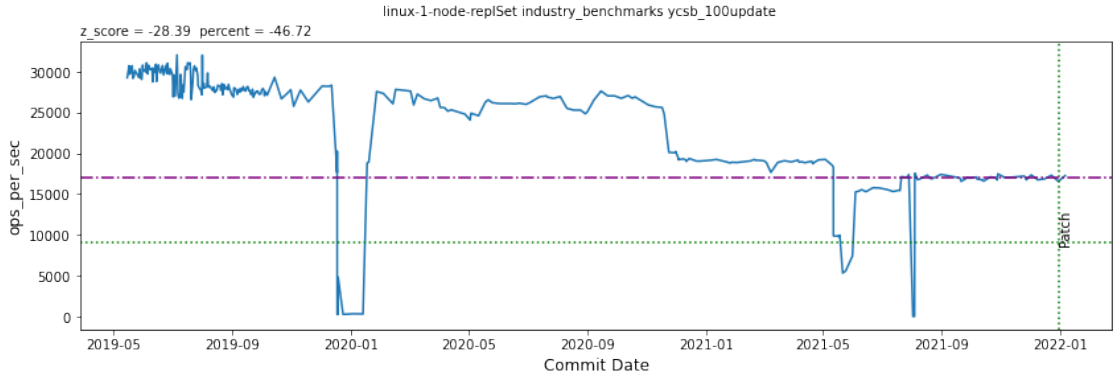
[13]:
```
for i in range(len(unsafe)):
    plot_timeseries(i)
    print(RenderHyperlink("link", "https://evergreen.mongodb.com/task/" +␣
 ↪unsafe.at[i,'task_id']))
```



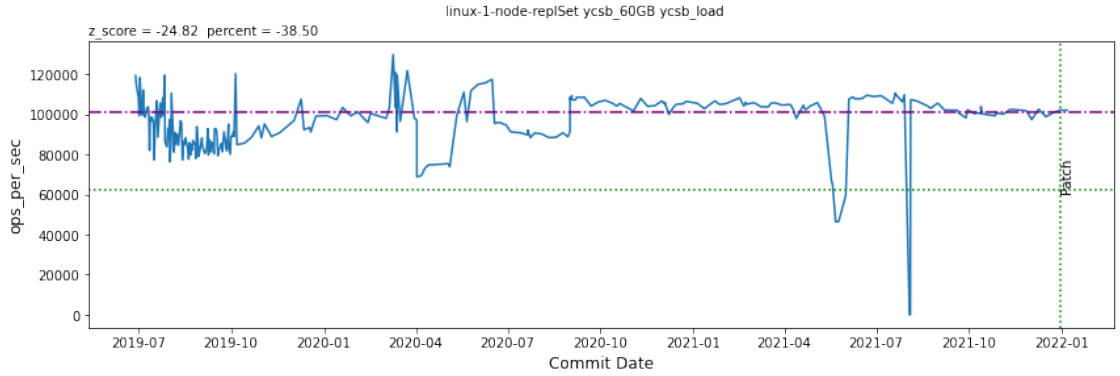
<IPython.core.display.HTML object>



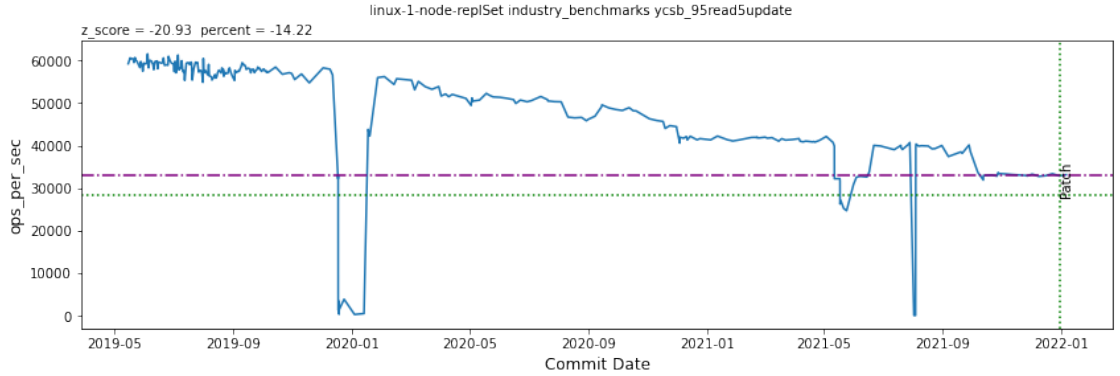
<IPython.core.display.HTML object>



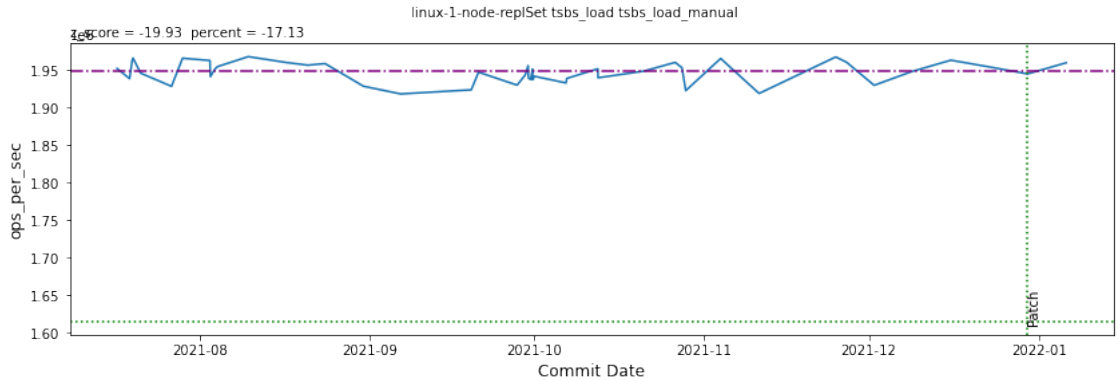
<IPython.core.display.HTML object>



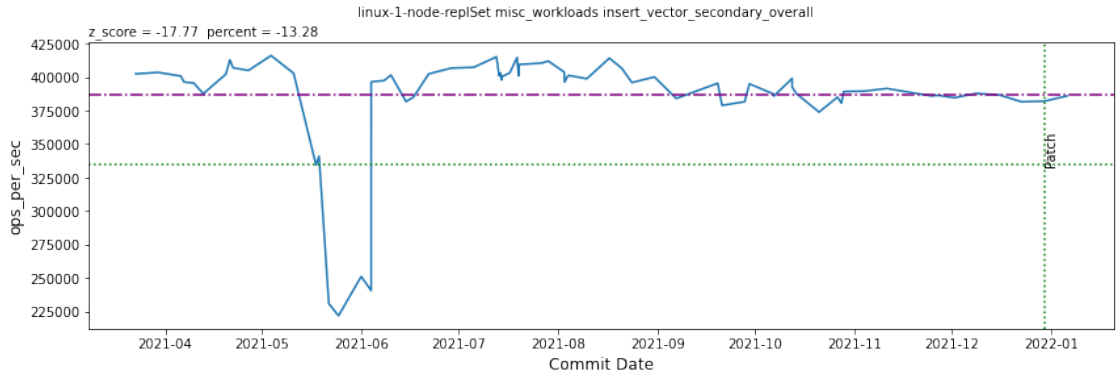
<IPython.core.display.HTML object>



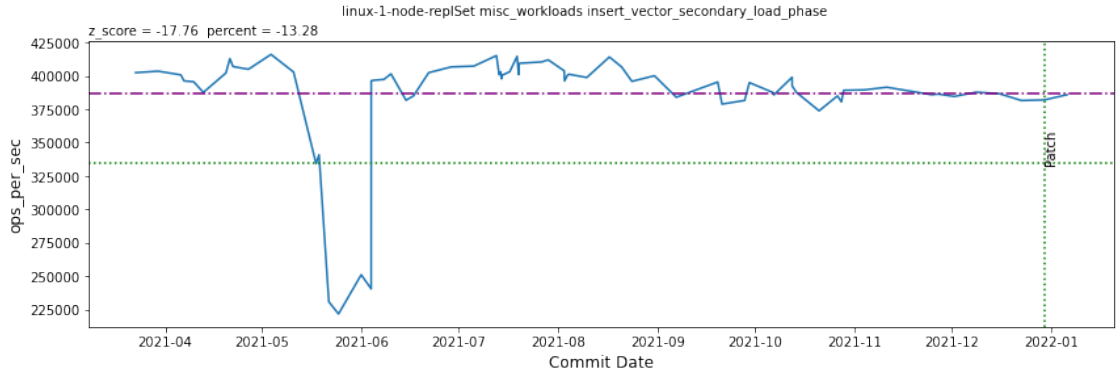
<IPython.core.display.HTML object>



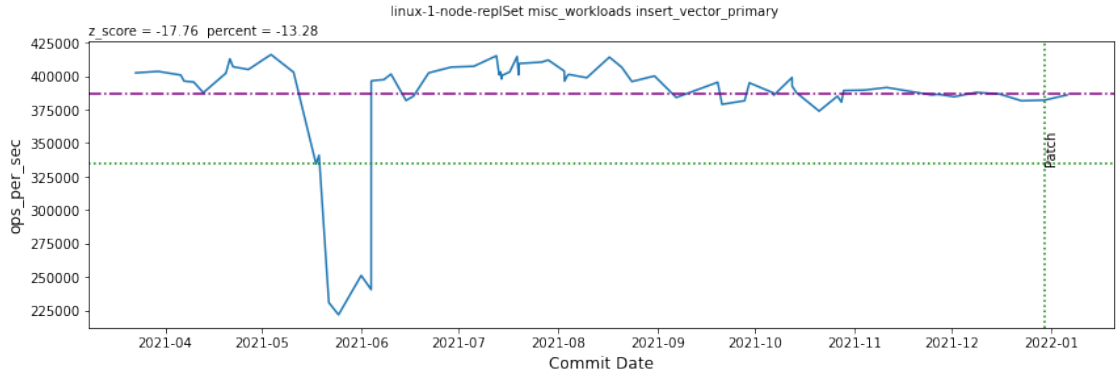
<IPython.core.display.HTML object>



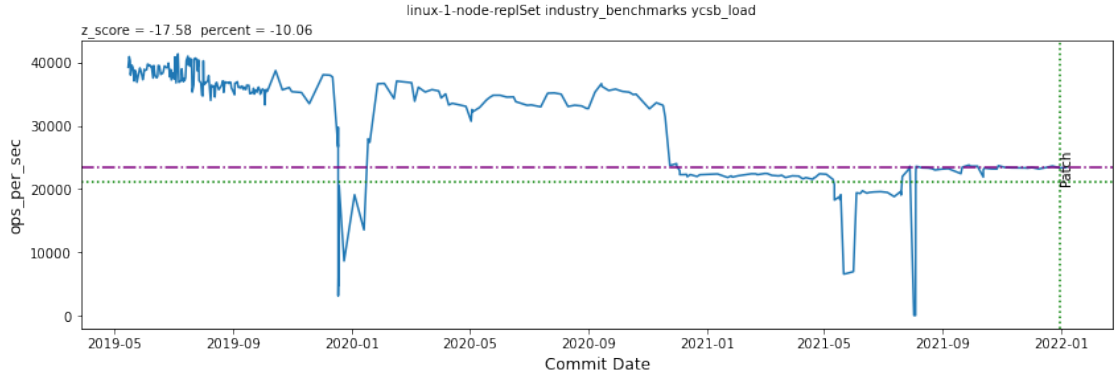
<IPython.core.display.HTML object>



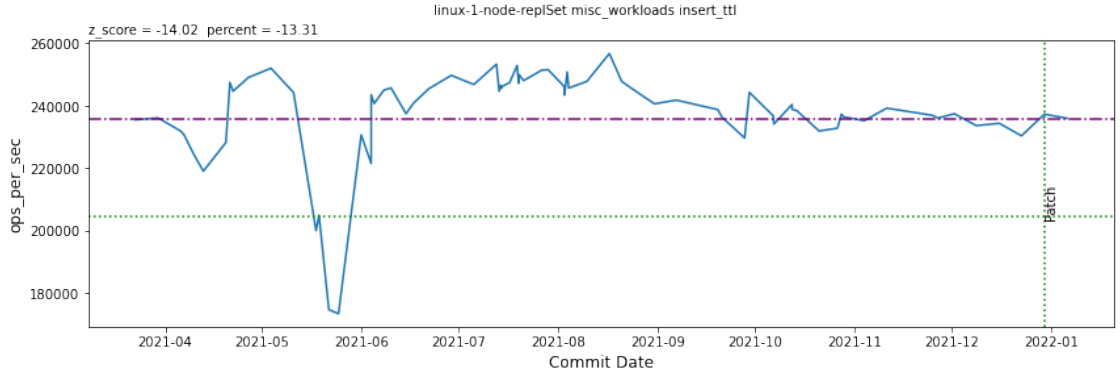
<IPython.core.display.HTML object>



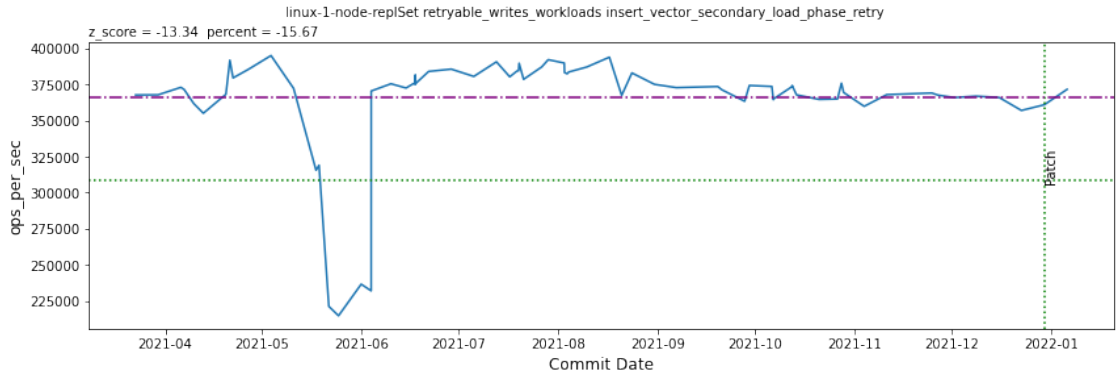
<IPython.core.display.HTML object>



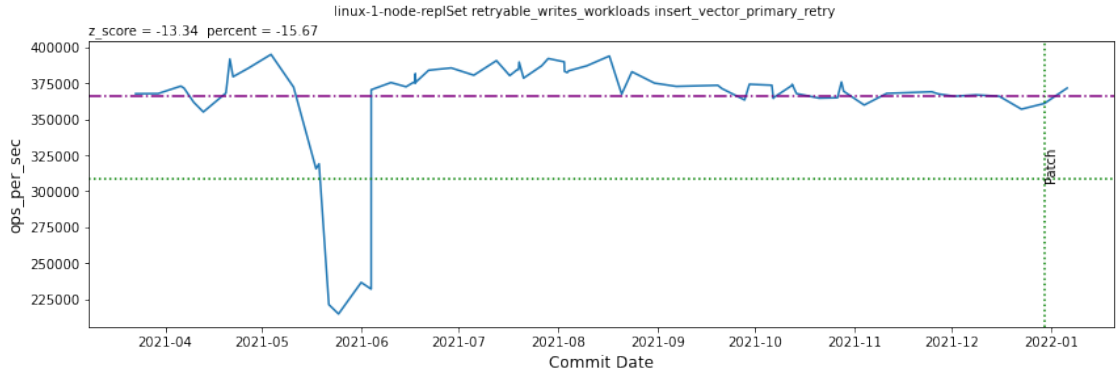
<IPython.core.display.HTML object>



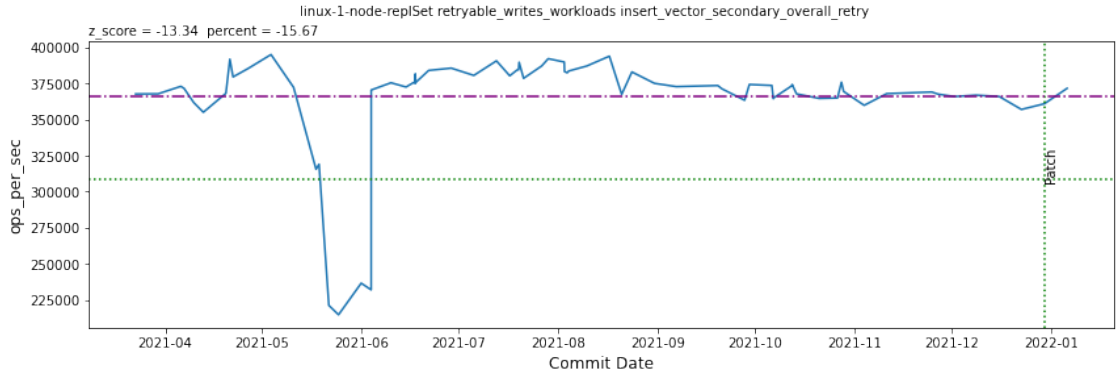
<IPython.core.display.HTML object>



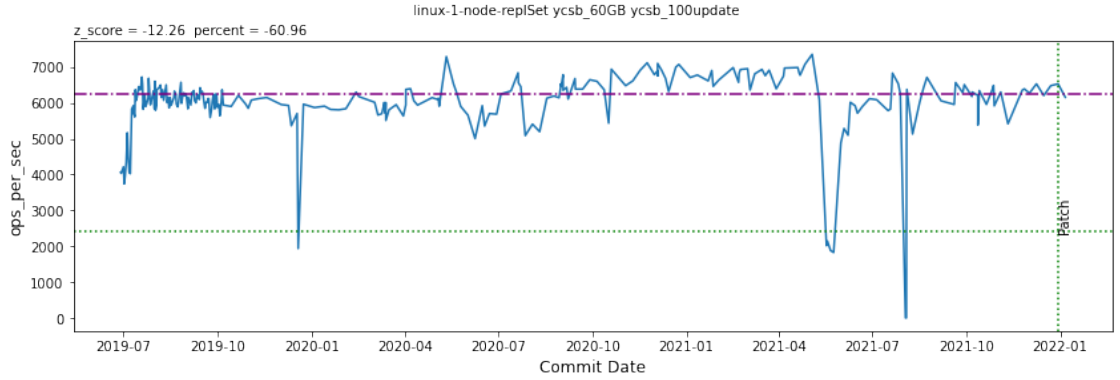
<IPython.core.display.HTML object>



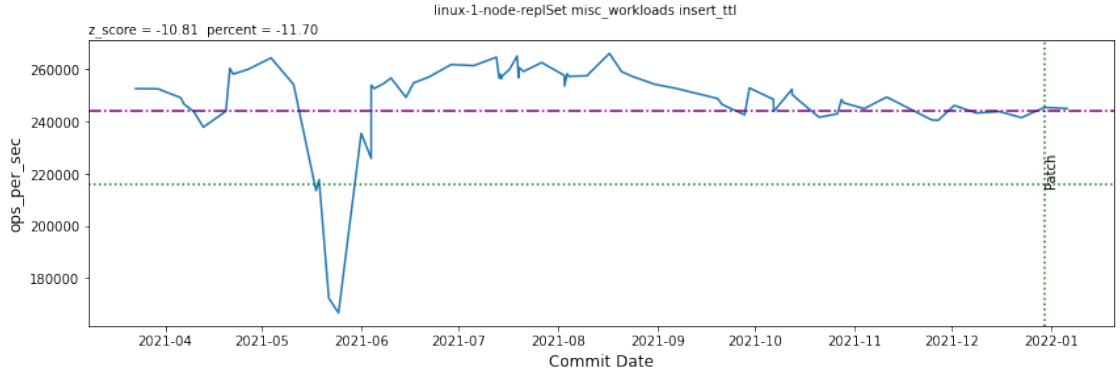
<IPython.core.display.HTML object>



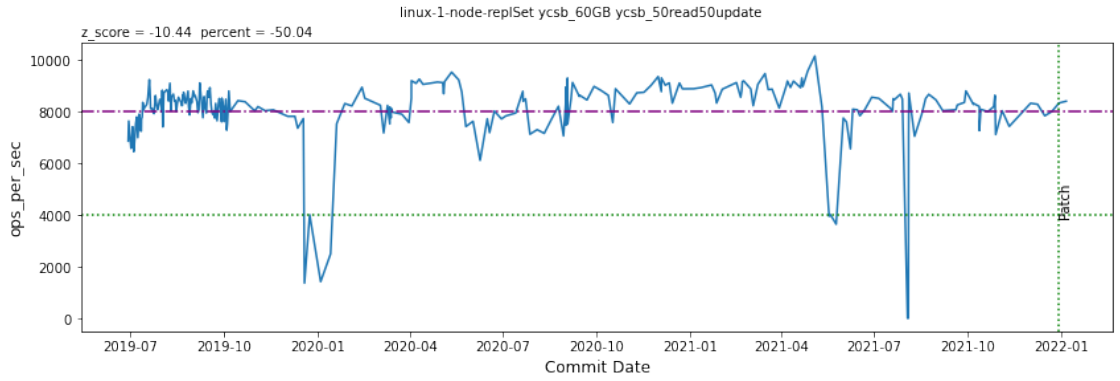
<IPython.core.display.HTML object>



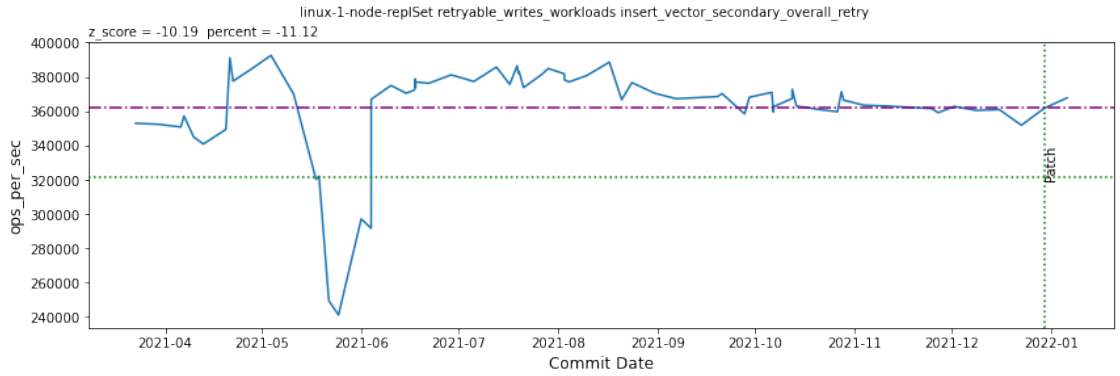
<IPython.core.display.HTML object>



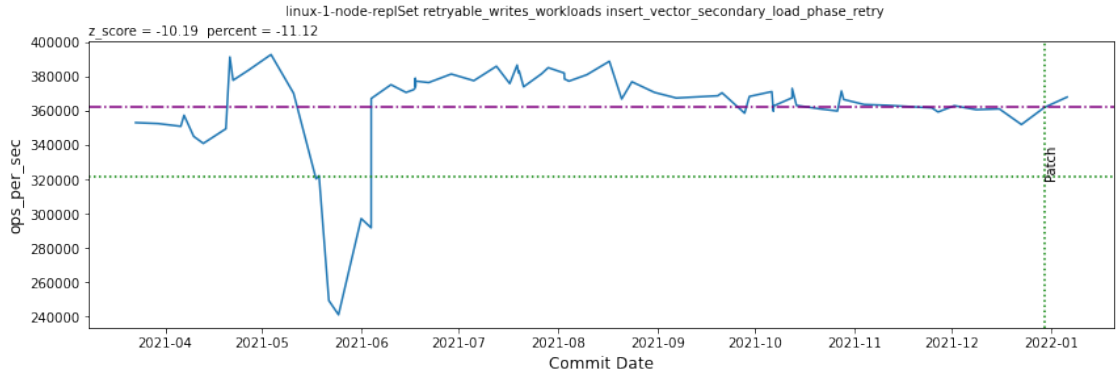
<IPython.core.display.HTML object>



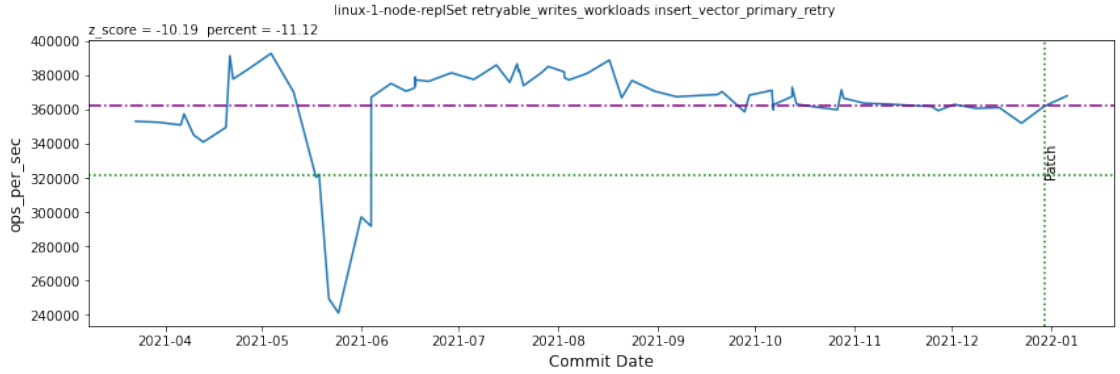
<IPython.core.display.HTML object>



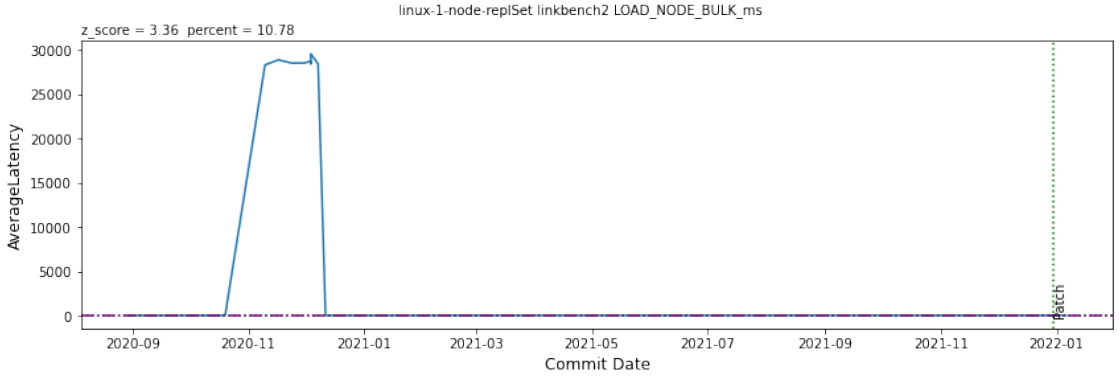
<IPython.core.display.HTML object>



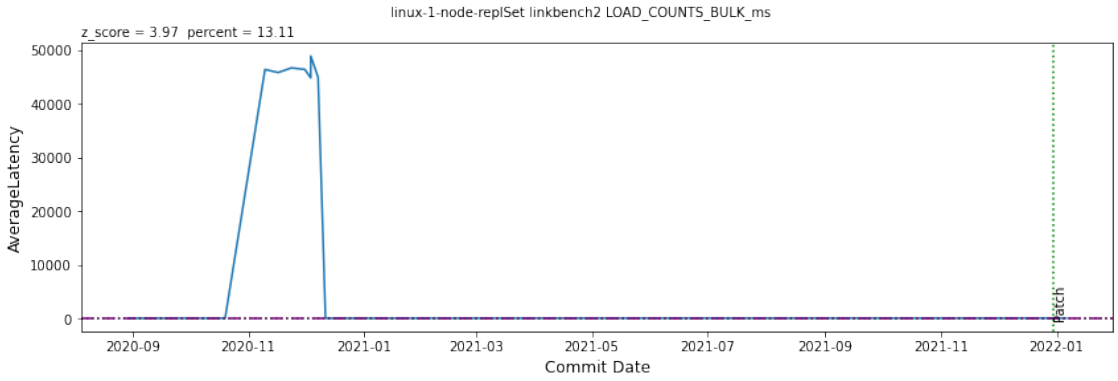
<IPython.core.display.HTML object>



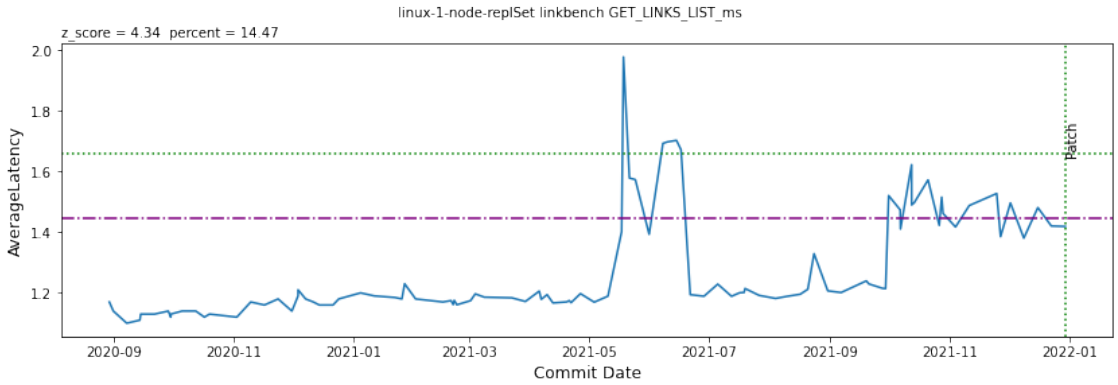
<IPython.core.display.HTML object>



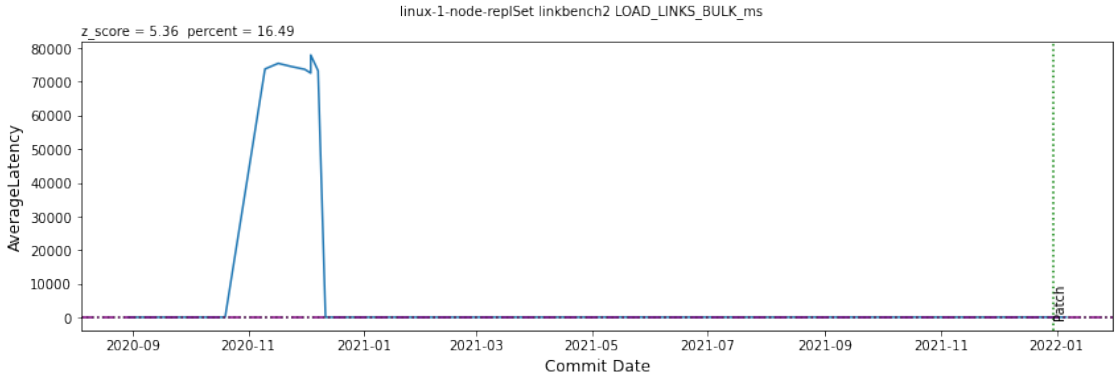
<IPython.core.display.HTML object>



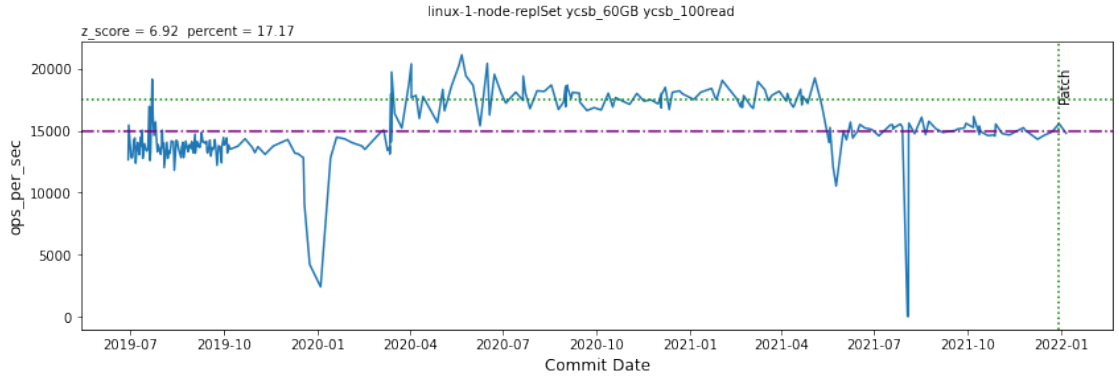
<IPython.core.display.HTML object>



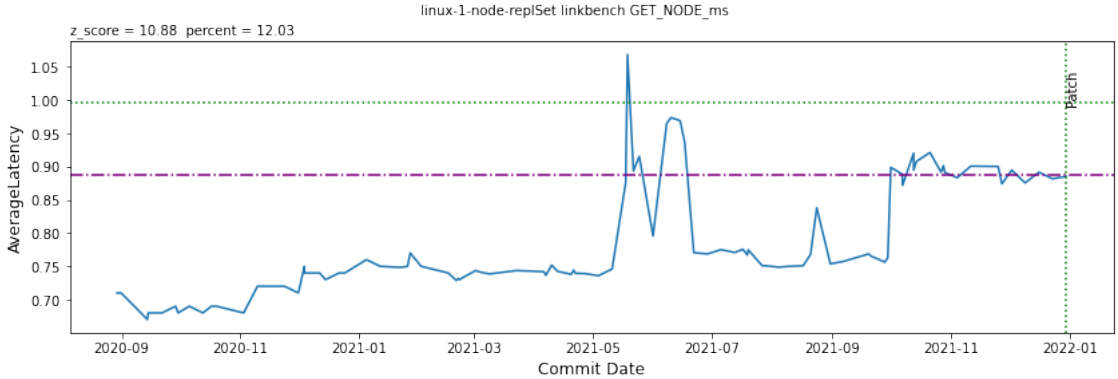
<IPython.core.display.HTML object>



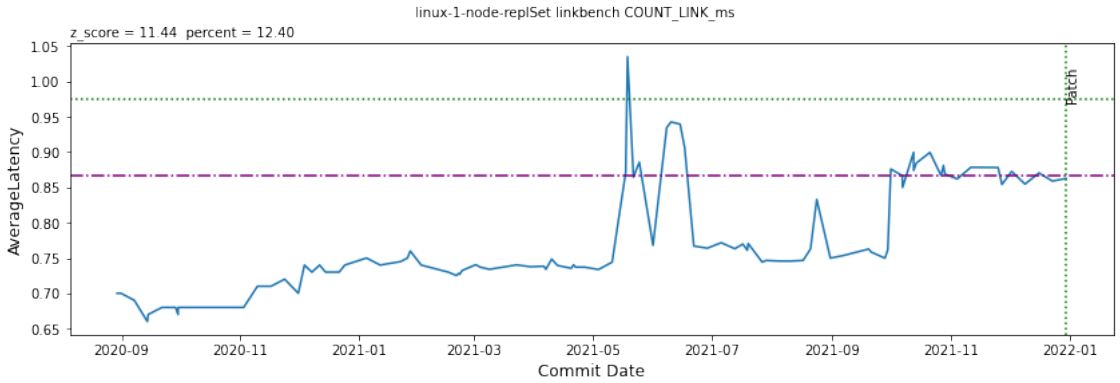
<IPython.core.display.HTML object>



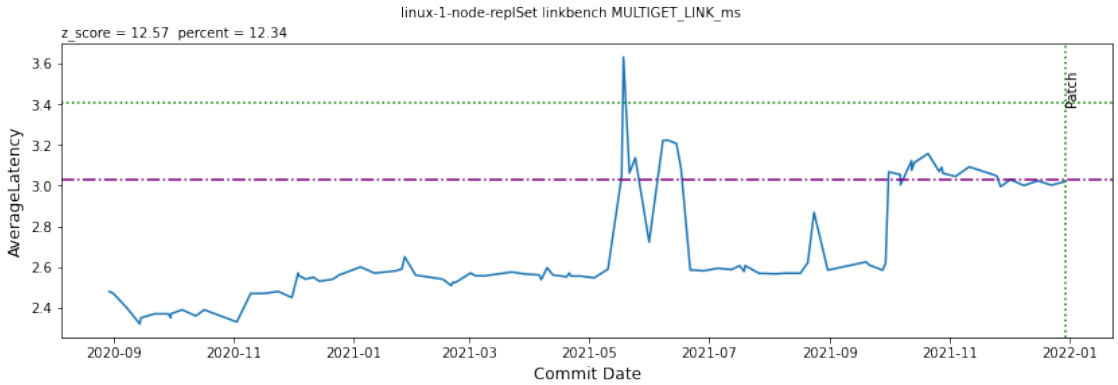
<IPython.core.display.HTML object>



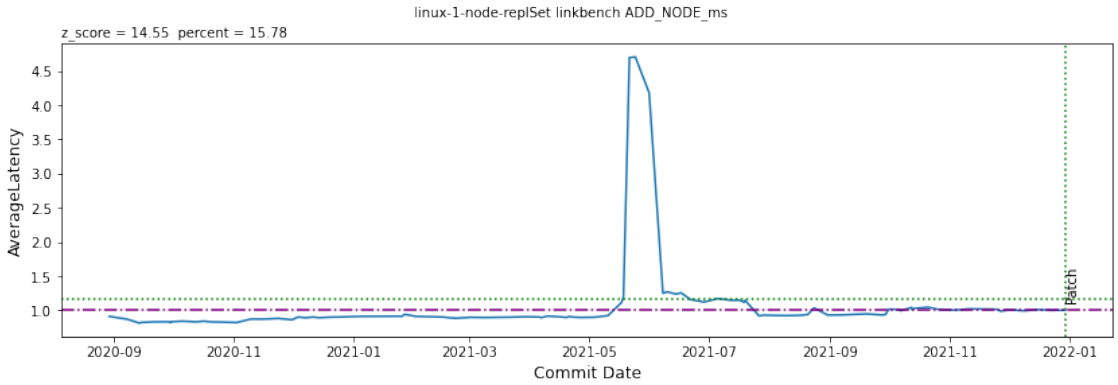
<IPython.core.display.HTML object>



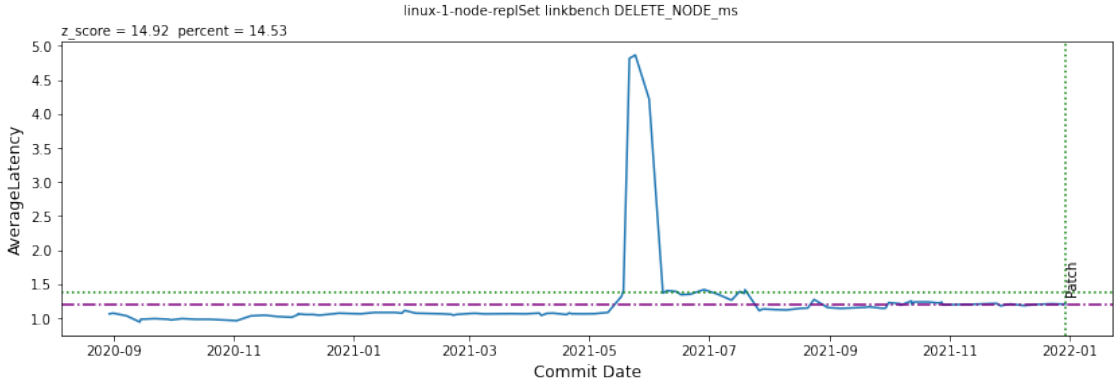
<IPython.core.display.HTML object>



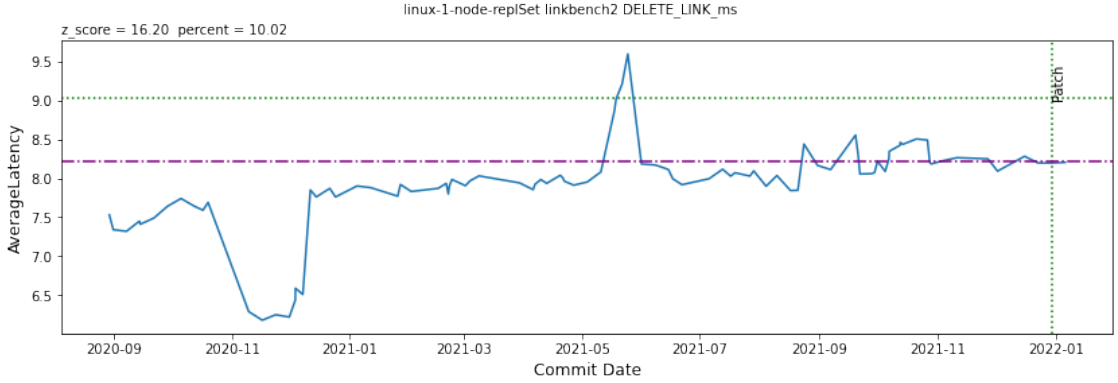
<IPython.core.display.HTML object>



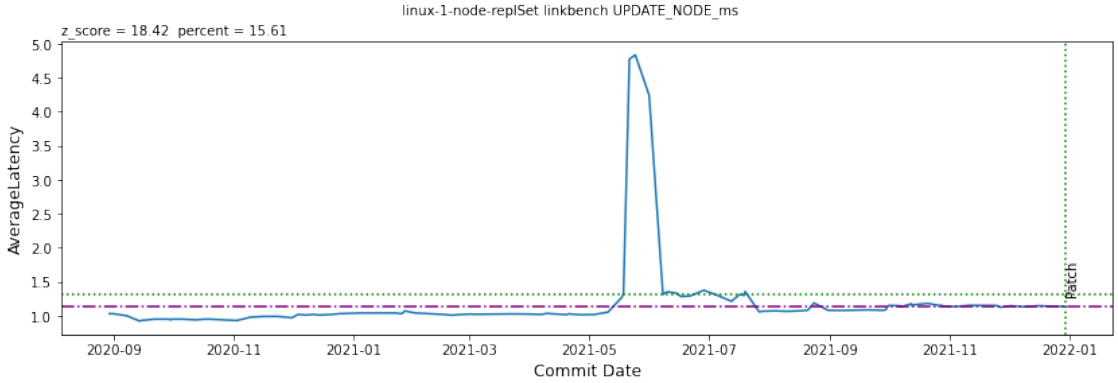
<IPython.core.display.HTML object>



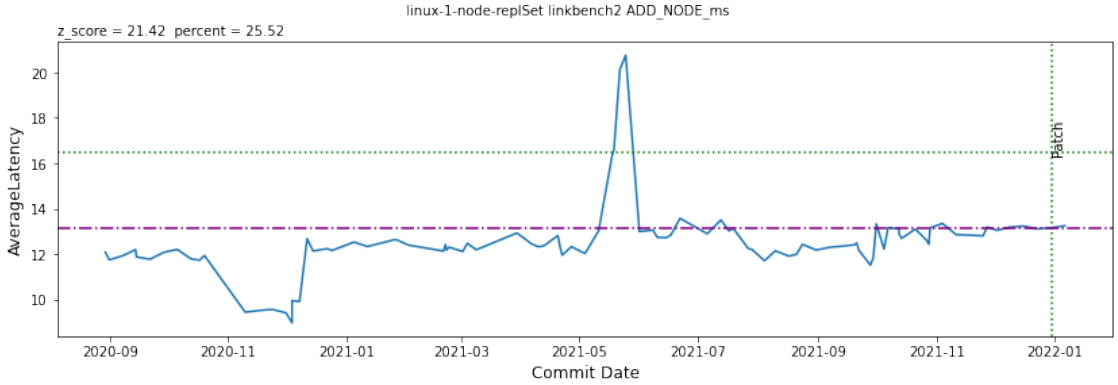
<IPython.core.display.HTML object>



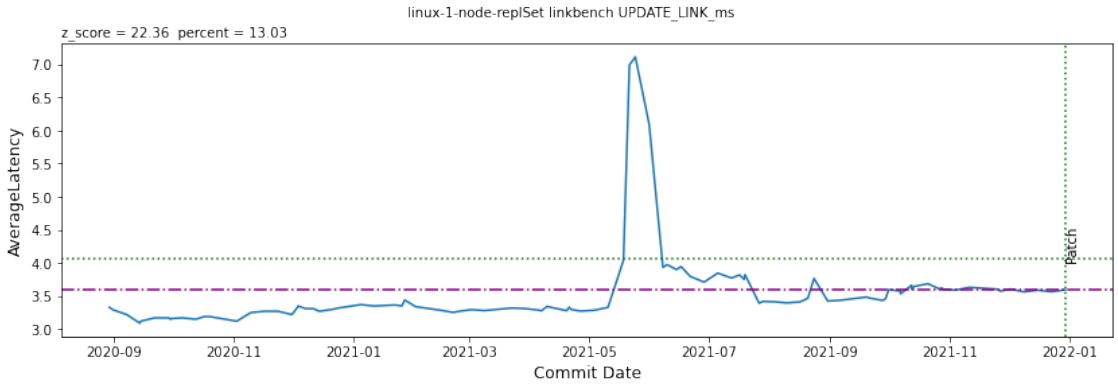
<IPython.core.display.HTML object>



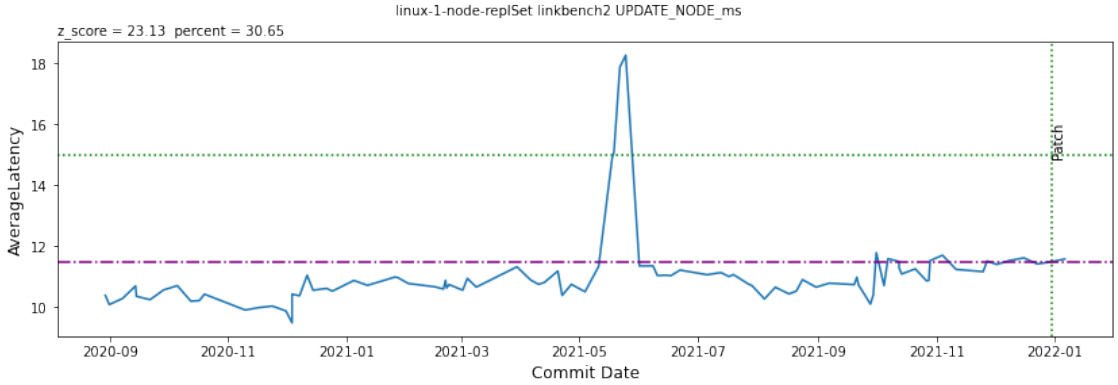
<IPython.core.display.HTML object>



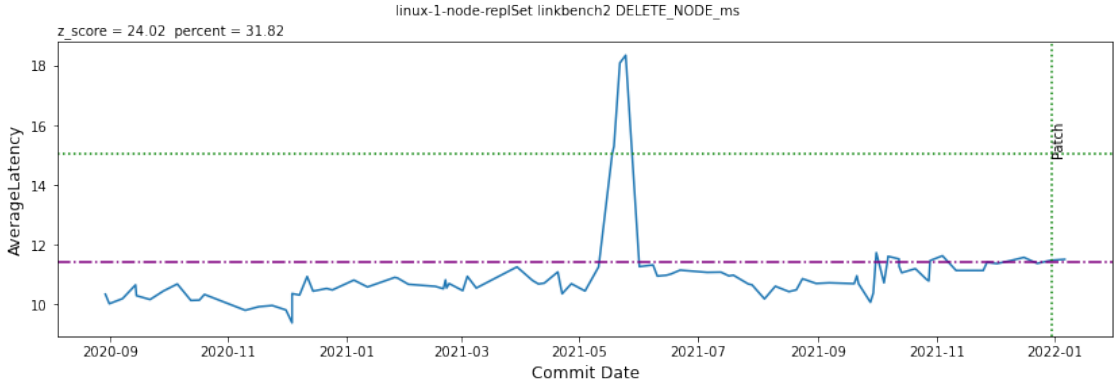
<IPython.core.display.HTML object>



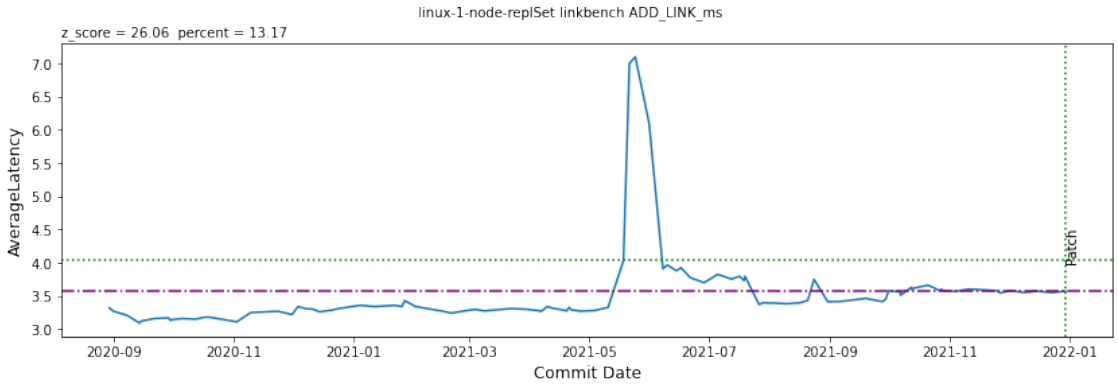
<IPython.core.display.HTML object>



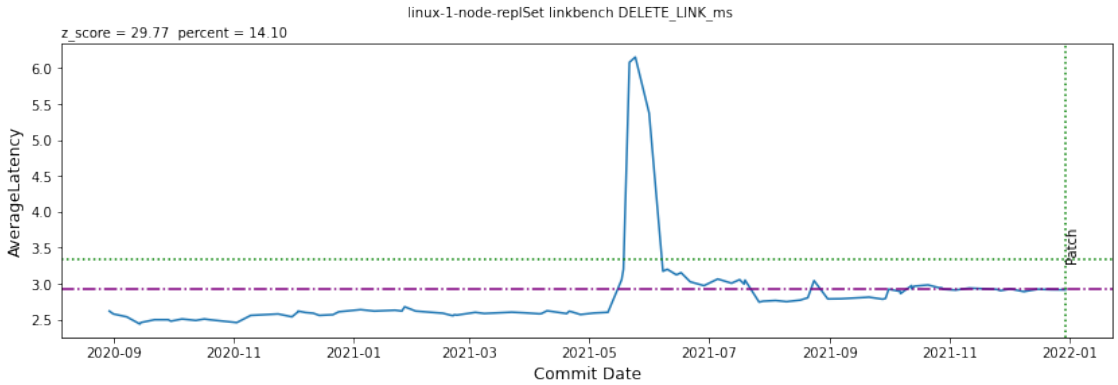
<IPython.core.display.HTML object>



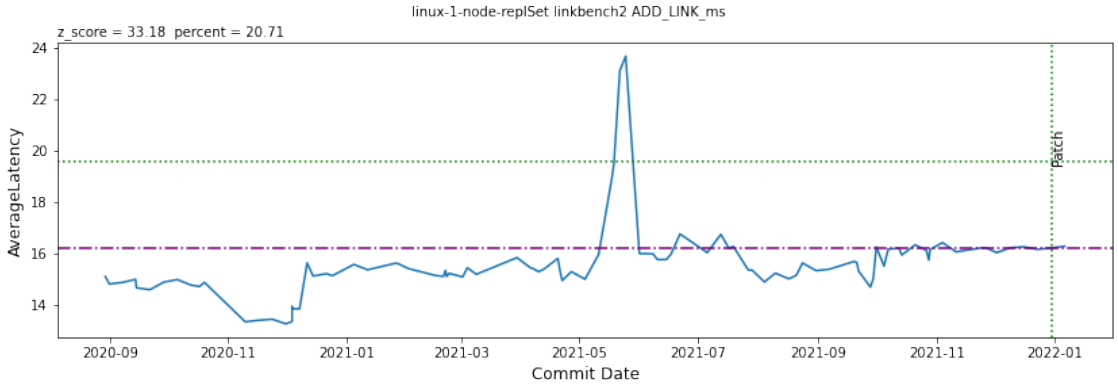
<IPython.core.display.HTML object>



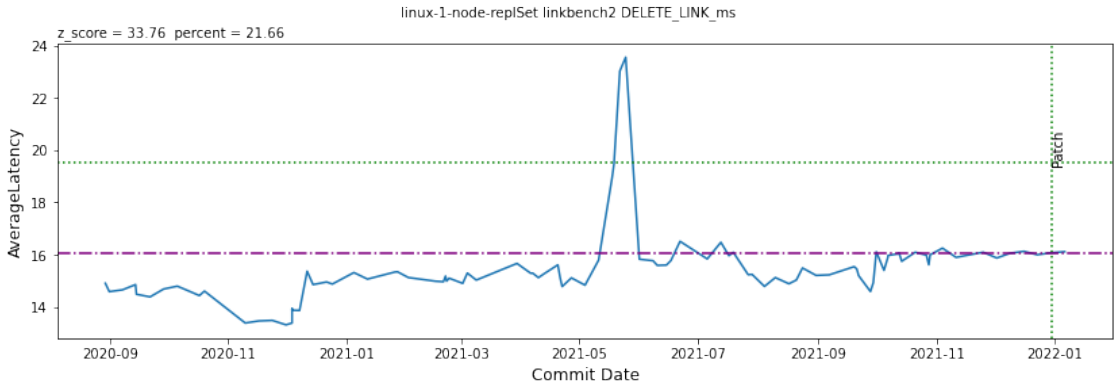
<IPython.core.display.HTML object>



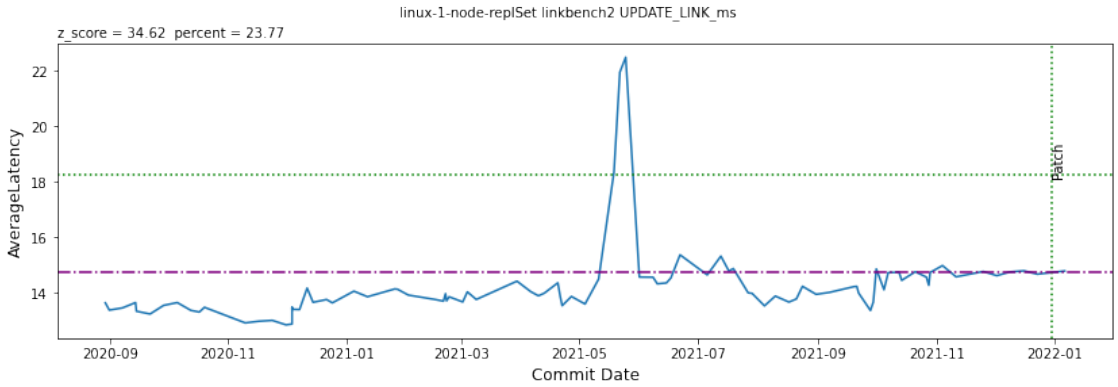
<IPython.core.display.HTML object>



<IPython.core.display.HTML object>



<IPython.core.display.HTML object>



```
<IPython.core.display.HTML object>
```

[ ]: