Scale Features and Build Model

Scales Raw Features

Import CSV of Aggregated Darshan Logs
Apply Log10 and Percent Scaling

In [1]:
```
import os
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
```

In [2]:
```
df = pd.read_csv("./raws.csv",lineterminator='\n',sep = ',' ,error_bad_line
#df.mean()
```

In [3]:
```
df = df.drop(df.columns[0],axis = 1)
df = df.drop(df.columns[0],axis = 1)
f = pd.DataFrame()
```

In [4]:
```
df
```

Out[4]:

	posix_read_time	posix_write_time	posix_meta_time	posix_bytes_read	posix_bytes_read_100
0	0.000000	0.000000	0.000000	0.000000e+00	0.0
1	104.611641	10.024055	20.060841	2.390891e+10	147688.0
2	124.560730	42.051125	54.839272	5.019637e+10	332059.0
3	25763.292969	582.297363	24.895737	5.488943e+12	30785.0
4	154.534821	681.548279	658.484985	2.293203e+10	588029.0
...	...	...	...	...	...
875282	138.354477	82.278084	194.485565	5.593977e+10	216146.0
875283	54.443073	231.440857	25.271391	1.465277e+09	3099.0
875284	0.000000	0.000000	0.000000	0.000000e+00	0.0
875285	0.000000	0.000000	0.000000	0.000000e+00	0.0
875286	227.063828	191.747269	172.671997	1.077412e+11	775359.0

875287 rows × 50 columns

In [5]:

```
df = df.dropna(axis=0, how='any')
df.columns
```

Out[5]:

```
Index(['posix_read_time', 'posix_write_time', 'posix_meta_time',
       'posix_bytes_read', 'posix_bytes_read_100', 'posix_bytes_read_1K',
       'posix_bytes_read_10K', 'posix_bytes_read_100K', 'posix_bytes_read
_11M',
       'posix_bytes_read_4M', 'posix_bytes_read_10M', 'posix_bytes_read_1
00M',
       'posix_bytes_read_1G', 'posix_bytes_read_PLUS', 'posix_bytes_writ
e',
       'posix_bytes_write_100', 'posix_bytes_write_1K',
       'posix_bytes_write_10K', 'posix_bytes_write_100K',
       'posix_bytes_write_1M', 'posix_bytes_write_4M', 'posix_bytes_write
_10M',
       'posix_bytes_write_100M', 'posix_bytes_write_1G',
       'posix_bytes_write_PLUS', 'posix_opens', 'posix_reads', 'posix_wri
tes',
       'posix_seeks', 'posix_stats', 'posix_mmaps', 'posix_fsyncs',
       'posix_fdsyncs', 'posix_rename_sources', 'posix_rename_targets',
       'posix_renamed_from', 'posix_renamed_mode', 'posix_number_of_file
s',
       'nprocs', 'posix_f_align', 'posix_m_align', 'lustre_number_of_file
s',
       'lustre_mdts', 'lustre_osts', 'lustre_stripe_size',
       'lustre_stripe_offset', 'lustre_stripe_width', 'lustre_number_of_o
sts',
       'jobid', 'path'],
      dtype='object')
```

In [6]:

```
#files
f['log10_p_files'] = df['posix_number_of_files']
f['log10_l_files'] = df['lustre_number_of_files']
```

In [7]:

```
#accesses
df['p_accesses'] = df['posix_reads'] + df['posix_writes']
f['log10_p_accesses'] = df['p_accesses']

f['log10_p_accesses']
```

Out[7]:

```
0               0.0
1          880136.0
2         2379598.0
3         8903411.0
4         7846387.0
            ...
875282    2234152.0
875283     197651.0
875284          0.0
875285          0.0
875286    6065006.0
Name: log10_p_accesses, Length: 875287, dtype: float64
```

In [8]:
```
#bytes
f['p_bytes'] = df['posix_bytes_read']
```

In [9]:
```
f['p_opens'] = df['posix_opens']
f['p_seeks'] = df['posix_seeks']
f['p_stats'] = df['posix_stats']
f['p_mode'] = df['posix_renamed_mode']
```

In [10]:
```
f['l_n_osts'] = df['lustre_number_of_osts']
f['l_stripe_w'] = df['lustre_stripe_width']
f['l_mdts'] = df['lustre_mdts']
```

In [11]:
```
f['log10_p_nprocs'] = df['nprocs']
f['log10_p_falign'] = df['posix_f_align']
f['log10_p_malign'] = df['posix_m_align']
```

In [12]:
```
f['perc_p_reads'] = df['posix_reads']
f['perc_p_writes'] = df['posix_writes']
```

In [13]:
```
f['perc_p_bytes_read_100'] = df['posix_bytes_read_100']
f['perc_p_bytes_read_1K'] = df['posix_bytes_read_1K']
f['perc_p_bytes_read_10K'] = df['posix_bytes_read_10K']
f['perc_p_bytes_read_100K'] = df['posix_bytes_read_100K']
f['perc_p_bytes_read_1M'] = df['posix_bytes_read_11M']
f['perc_p_bytes_read_4M'] = df['posix_bytes_read_4M']
f['perc_p_bytes_read_10M'] = df['posix_bytes_read_10M']

f['perc_p_bytes_read_100M'] = df['posix_bytes_read_100M']
f['perc_p_bytes_read_1G'] = df['posix_bytes_read_1G']
f['perc_p_bytes_read_PLUS'] = df['posix_bytes_read_PLUS']
```

In [14]:
```
f['perc_p_bytes_write_100'] = df['posix_bytes_write_100']
f['perc_p_bytes_write_1K'] = df['posix_bytes_write_1K']
f['perc_p_bytes_write_10K'] = df['posix_bytes_write_10K']
f['perc_p_bytes_write_100K'] = df['posix_bytes_write_100K']
f['perc_p_bytes_write_1M'] = df['posix_bytes_write_1M']
f['perc_p_bytes_write_4M'] = df['posix_bytes_write_4M']
f['perc_p_bytes_write_10M'] = df['posix_bytes_write_10M']
f['perc_p_bytes_write_100M'] = df['posix_bytes_write_100M']
f['perc_p_bytes_write_1G'] = df['posix_bytes_write_1G']
f['perc_p_bytes_write_PLUS'] = df['posix_bytes_write_PLUS']


f = f.replace(-np.inf, -1)
f = f.replace(np.nan, 0)
```

In [15]:
```
df['time'] = df['posix_write_time'].astype('float') + df['posix_read_time']
```

In [16]:
```
df['bytes'] = df['posix_bytes_read'].astype('float') + df['posix_bytes_writ
```

In [17]:
```
#df = df[df['bytes'] >9999999]
```

In [18]:

```
f['throughput'] = df['bytes'].astype('float') / df['time']

f = f[f['throughput'] >0]
```

In [19]:

```
#delete columns with all zeros
f = f.loc[:, (f != 0).any(axis=0)]

#remove infinite values
f = f.replace([np.inf, -np.inf], np.nan).dropna(axis=0)

f.max()
```

Out[19]:

```
log10_p_files              1.219270e+05
log10_l_files              1.219270e+05
log10_p_accesses           2.251942e+10
p_bytes                    3.038456e+14
p_opens                    5.531094e+08
p_seeks                    1.445220e+10
p_stats                    6.522921e+07
p_mode                     5.337293e+07
l_n_osts                   3.600000e+02
l_stripe_w                 7.438575e+06
l_mdts                     1.000000e+00
log10_p_nprocs             3.520000e+05
log10_p_falign             1.422540e+11
log10_p_malign             1.085312e+06
perc_p_reads               2.237846e+10
perc_p_writes              1.302770e+10
perc_p_bytes_read_100      5.221517e+08
perc_p_bytes_read_1K       2.074657e+10
perc_p_bytes_read_10K      1.536278e+09
perc_p_bytes_read_100K     1.515506e+08
perc_p_bytes_read_1M       4.044503e+08
perc_p_bytes_read_4M       6.561462e+07
perc_p_bytes_read_10M      2.083200e+06
perc_p_bytes_read_100M     2.872090e+05
perc_p_bytes_read_1G       1.792000e+06
perc_p_bytes_write_100     1.302770e+10
perc_p_bytes_write_1K      2.852127e+09
perc_p_bytes_write_10K     3.867477e+08
perc_p_bytes_write_100K    8.347452e+07
perc_p_bytes_write_1M      1.357245e+07
perc_p_bytes_write_4M      3.839488e+06
perc_p_bytes_write_10M     6.190660e+05
perc_p_bytes_write_100M    1.249280e+06
perc_p_bytes_write_1G      1.937500e+04
throughput                 2.344536e+09
dtype: float64
```

In [20]:
```
t = pd.DataFrame()
t['throughput'] = f['throughput']
f = f.drop(labels = 'throughput', axis = 1)
f
```

Out[20]:

	log10_p_files	log10_l_files	log10_p_accesses	p_bytes	p_opens	p_seeks	p_stats
1	799.0	176.0	880136.0	2.390891e+10	8858.0	319241.0	34901.0
2	360.0	224.0	2379598.0	5.019637e+10	62398.0	1107764.0	270222.0
3	290.0	290.0	8903411.0	5.488943e+12	8711.0	2010273.0	28432.0
4	319.0	201.0	7846387.0	2.293203e+10	23158.0	6015926.0	400399.0
6	428.0	190.0	6647935.0	5.209185e+10	69261.0	4608438.0	410846.0
...	...	...	...	...	...	...	...
875280	1.0	1.0	57344.0	3.006477e+10	1808.0	59128.0	12.0
875281	13.0	4.0	102439.0	7.467916e+08	193.0	91.0	385.0
875282	624.0	124.0	2234152.0	5.593977e+10	35112.0	1035457.0	159180.0
875283	1088.0	1088.0	197651.0	1.465277e+09	2112.0	12509.0	4198.0
875286	582.0	128.0	6065006.0	1.077412e+11	119432.0	2750807.0	578508.0

671063 rows × 34 columns

In [21]:
```
df = df[df.index.isin(t.index)]
t = t.reset_index()
f = f.reset_index()
f = f.drop(f.columns[0] , axis =1)
t = t.drop(t.columns[0] , axis =1)
```

In [22]:
```
f = StandardScaler().fit_transform(f)
```

In [23]:
```
t
```

Out[23]:

	throughput
0	1.803194e+08
1	2.282342e+08
2	2.083669e+08
3	1.724841e+07
4	4.581690e+07
...	...
671058	5.482767e+08
671059	2.136637e+07
671060	1.362498e+08
671061	1.942413e+07
671062	1.842942e+08

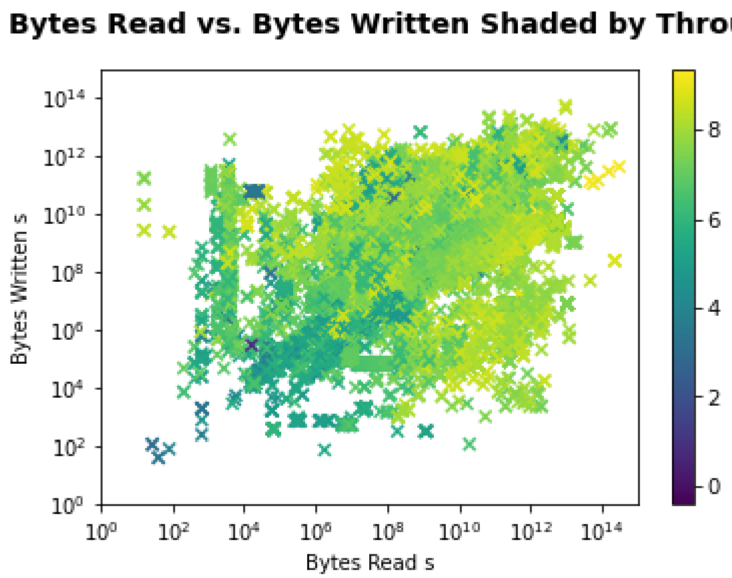
671063 rows × 1 columns

In [24]:
```
print(t.min())
print(t.max())
```

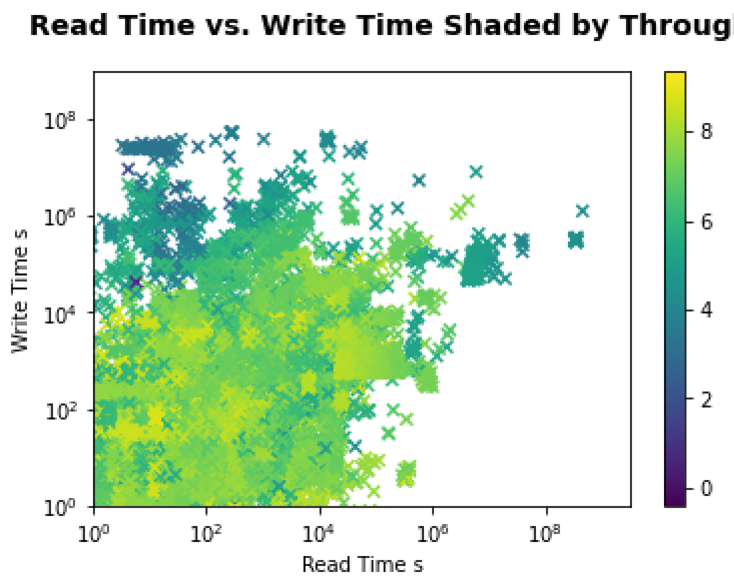
```
throughput    0.39201
dtype: float64
throughput    2.344536e+09
dtype: float64
```

In [25]:
```
rseed = 1
t_size = 0.4
train_data, test_data, train_labels, test_labels = train_test_split(f,t, te
```

In [26]:
```
fig = plt.figure()
fig.suptitle('Bytes Read vs. Bytes Written Shaded by Throughput', fontsize=

ax = fig.add_subplot(111)
sp = ax.scatter(df['posix_bytes_read'],df['posix_bytes_write'], marker = 'x

ax.set_xlabel('Bytes Read s')
ax.set_ylabel('Bytes Written s')
ax.loglog()
#plt.autoscale(enable=True, axis='y')
plt.xlim(10**0,10**15)
plt.ylim(10**0,10**15)
fig.colorbar(sp)
plt.show()
```



In [27]:
```
fig = plt.figure()
fig.suptitle('Read Time vs. Write Time Shaded by Throughput', fontsize=14,

ax = fig.add_subplot(111)
sp = ax.scatter(df['posix_read_time'],df['posix_write_time'], marker = 'x',

ax.set_xlabel('Read Time s')
ax.set_ylabel('Write Time s')
ax.loglog()
#plt.autoscale(enable=True, axis='y')
plt.xlim(10**0,10**9.5)
plt.ylim(10**0,10**9)
fig.colorbar(sp)
plt.show()
```



In [28]:
```
from sklearn.cluster import KMeans

k = 3
# Create a KMeans instance with k clusters: model
model = KMeans(n_clusters=k,max_iter = 20**10,random_state=rseed)

# Fit model to samples
model.fit(f)

cluster_labels = model.predict(f)
```

In [29]:
```
print('c0',(cluster_labels ==0).sum())
print('c1',(cluster_labels ==1).sum())
print('c2',(cluster_labels ==2).sum())
print('c3',(cluster_labels ==3).sum())
print('c4',(cluster_labels ==4).sum())
print('c5',(cluster_labels ==5).sum())
print(t.shape)
#How many items in each cluster
```

```
c0 670371
c1 678
c2 14
c3 0
c4 0
c5 0
(671063, 1)
```

In [30]:
```
#cluster splits
t5 = t[pd.Series((cluster_labels == 5).tolist()).astype('bool')]
t4 = t[pd.Series((cluster_labels == 4).tolist()).astype('bool')]
t3 = t[pd.Series((cluster_labels == 3).tolist()).astype('bool')]
t2 = t[pd.Series((cluster_labels == 2).tolist()).astype('bool')]
t1 = t[pd.Series((cluster_labels == 1).tolist()).astype('bool')]
t0 = t[pd.Series((cluster_labels == 0).tolist()).astype('bool')]

f5 = f[pd.Series((cluster_labels == 5).tolist()).astype('bool')]
f4 = f[pd.Series((cluster_labels == 4).tolist()).astype('bool')]
f3 = f[pd.Series((cluster_labels == 3).tolist()).astype('bool')]
f2 = f[pd.Series((cluster_labels == 2).tolist()).astype('bool')]
f1 = f[pd.Series((cluster_labels == 1).tolist()).astype('bool')]
f0 = f[pd.Series((cluster_labels == 0).tolist()).astype('bool')]
```

In [31]:
```
reaggregated_predictions = pd.DataFrame()
reaggregated_truths = pd.DataFrame()
print(reaggregated_predictions.shape)
print(reaggregated_truths.shape)
```

```
(0, 0)
(0, 0)
```

In [ ]:

In [32]:
```
import xgboost as xg
train_data, test_data, train_labels, test_labels = train_test_split(f0,t0,
xgb_r = xg.XGBRegressor(n_estimators = 10000, seed = 123)
print(train_labels)
xgb_r.fit(train_data, train_labels)
predicted_labels = xgb_r.predict(test_data)


print(reaggregated_predictions.shape)
print(reaggregated_truths.shape)
```

```
          throughput
41044   3.216501e+08
64936   5.062943e+07
232553  3.757794e+08
471110  4.285667e+08
59102   1.131367e+07
...              ...
371813  2.306442e+07
491780  1.711825e+07
471423  9.933681e+06
492272  3.768211e+07
128179  8.902062e+06

[402222 rows x 1 columns]
(0, 0)
(0, 0)
```

In [33]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_log_error
print("Mean True Value: \t",int(test_labels.mean() ))
print("Mean Absolute Error: \t", int(mean_absolute_error(test_labels, predi
print("Mean Squared Error: ", mean_squared_error(test_labels, predicted_lab
print("Root Mean Squared Error: ", mean_squared_error(test_labels, predicte

#print("Mean Squared Logarithmic Error : ", mean_squared_log_error( test_la
from sklearn.metrics import mean_absolute_percentage_error
print("MAPE :" + str(mean_absolute_percentage_error( test_labels, predicted

print("R2: " + str(r2_score(test_labels,predicted_labels)) + "\n")

print(xgb_r.feature_importances_)
```

```
Mean True Value:         84333781
Mean Absolute Error:     17245685
Mean Squared Error:  1270401411313262.2
Root Mean Squared Error:  35642690.853992246
MAPE :86.20154012391694
R2: 0.8926602252069487

[0.03221513 0.01281782 0.00302289 0.1191103  0.02911005 0.00926128
 0.00459579 0.01106033 0.00366708 0.03694115 0.         0.03296688
 0.00327673 0.         0.00902359 0.00967221 0.01884102 0.14283206
 0.00933851 0.00722012 0.05629914 0.0594423  0.00221861 0.00877629
 0.02907776 0.0038139  0.00709831 0.00674552 0.00486089 0.00653075
 0.12431251 0.04008806 0.01161676 0.14414619]
```

In [34]:
```
train_data, test_data, train_labels, test_labels = train_test_split(f1,t1,
xgb_r = xg.XGBRegressor(n_estimators = 10000, seed = 123)
xgb_r.fit(train_data, train_labels)
predicted_labels = xgb_r.predict(test_data)

reaggregated_predictions = pd.concat([reaggregated_predictions, pd.DataFram
reaggregated_truths = pd.concat([reaggregated_truths,pd.DataFrame(test_labe

print(reaggregated_predictions.shape)
print(reaggregated_truths.shape)

print(xgb_r.feature_importances_)
```

```
(272, 1)
(272, 1)
[1.54333073e-04 7.13654561e-03 6.45049789e-04 1.97636291e-01
 3.20529491e-02 4.52707559e-02 1.57306582e-04 8.85143309e-05
 2.18028857e-04 6.52674632e-03 0.00000000e+00 4.46196245e-05
 3.65037508e-02 0.00000000e+00 4.04543336e-03 8.97687860e-03
 8.48226901e-03 3.76282353e-03 1.05393445e-02 2.44787568e-03
 7.00901449e-03 2.21158680e-05 6.18716143e-03 5.95291676e-05
 1.84273722e-05 2.26261187e-03 5.16472086e-02 1.21423285e-02
 1.93237159e-02 1.96657027e-03 4.96727347e-01 1.02527079e-03
 1.15995517e-03 3.57591510e-02]
```

In [35]:
```
print("Mean Squared Error: ", mean_squared_error(test_labels, predicted_lab
print("Mean Absolute Error: ", mean_absolute_error(test_labels, predicted_l
print("Root Mean Squared Error: ", mean_squared_error(test_labels, predicte
#print("Mean Squared Logarithmic Error : ", mean_squared_log_error( test_la
print("MAPE :" + str(mean_absolute_percentage_error( test_labels, predicted
print("R2: " + str(r2_score(test_labels,predicted_labels)) + "\n")
```

```
Mean Squared Error:  332629147173365.9
Mean Absolute Error:  9922908.018586583
Root Mean Squared Error:  18238123.45537133
MAPE :3.6473689802098304
R2: 0.9043947630362277
```

In [36]:
```
train_data, test_data, train_labels, test_labels = train_test_split(f2,t2,
xgb_r = xg.XGBRegressor(n_estimators = 10000, seed = 123)
xgb_r.fit(train_data, train_labels)
predicted_labels = xgb_r.predict(test_data)
reaggregated_predictions = pd.concat([reaggregated_predictions, pd.DataFram
reaggregated_truths = pd.concat([reaggregated_truths,pd.DataFrame(test_labe

print("Mean Squared Error: ", mean_squared_error(test_labels, predicted_lab
print("Mean Absolute Error: ", mean_absolute_error(test_labels, predicted_l
print("Root Mean Squared Error: ", mean_squared_error(test_labels, predicte
#print("Mean Squared Logarithmic Error : ", mean_squared_log_error( test_la
print("MAPE :" + str(mean_absolute_percentage_error( test_labels, predicted
print("R2: " + str(r2_score(test_labels,predicted_labels)) + "\n")

print(reaggregated_predictions.shape)
print(reaggregated_truths.shape)

print(xgb_r.feature_importances_)
```

```
Mean Squared Error:  11224801543471.373
Mean Absolute Error:  1368860.494205026
Root Mean Squared Error:  3350343.496340543
MAPE :0.2021484466892188
R2: -0.20039406719512232

(278, 1)
(278, 1)
[0.03921963 0.03085561 0.         0.         0.         0.
 0.         0.         0.9299248  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
```

In [37]:
```
print("Mean Absolute Error: ", mean_absolute_error(reaggregated_truths, rea

print("Mean Squared Error: ", mean_squared_error(reaggregated_truths, reagg
print("MAPE :" + str(mean_absolute_percentage_error( reaggregated_truths, r
print("R2: " + str(r2_score(reaggregated_truths,reaggregated_predictions))

print(reaggregated_predictions.shape)
print(reaggregated_truths.shape)
```

```
Mean Absolute Error:  9738288.287844537
Mean Squared Error:  325692362735310.6
MAPE :3.5730117025079466
R2: 0.9050429876932115

(278, 1)
(278, 1)
```

In [ ]:

In [ ]:

In [38]:
```
e = reaggregated_predictions[0].values - reaggregated_truths['throughput'].
e
```

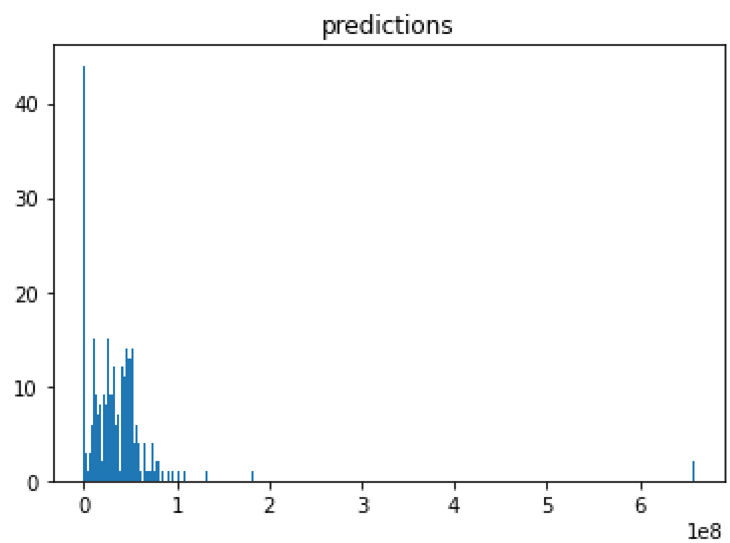
Out[38]:
```
array([ 9.32165267e+06, -1.39060502e+07, -4.30897915e+03, -2.34174507e+0
6,
        1.40894710e+06, -6.31435039e+06, -3.15716308e+03,  5.22114402e+0
5,
       -3.32085899e+05, -6.70552084e+06,  2.68573643e+07,  1.14117952e+0
7,
        3.09855980e+06,  3.74165425e+06,  2.52458508e+07, -4.82085938e+0
6,
       -4.45175026e+04, -3.04914398e+07,  4.05013985e+02, -1.74609644e+0
6,
       -8.96884041e+06, -1.80846091e+04, -1.76523806e+07,  3.52662018e+0
6,
        4.08376138e+07,  3.06673455e+07,  1.26002462e+07,  4.12626138e+0
7,
        4.46880650e+06,  1.11526214e+07,  2.80708421e+06, -9.87598291e+0
6,
       -1.66971191e+06, -6.09885535e+06, -6.38727361e+06,  8.75678249e+0
6,
        1.42505369e+07,  2.03249264e+06,  1.01096416e+06,  6.24631866e+0
5,
        3.66513242e+06, -7.47625887e+05, -2.51877973e+06,  2.98578129e+0
7,
        1.34692172e+06,  1.66097810e+07, -1.43881703e+07,  2.07066304e+0
4,
       -5.64434267e+06,  7.22954203e+06, -2.28182694e+06,  1.66034806e+0
4,
        6.44371649e+05,  2.57978226e+06, -9.81727522e+06,  9.92221784e+0
6,
        1.86282912e+07, -1.77388331e+07,  1.16040851e+05, -1.42867620e+0
6,
       -2.26166381e+06,  2.92945965e+03,  2.25561738e+05, -2.60384880e+0
7,
        2.74037491e+07,  9.00684812e+06,  1.76287372e+07, -6.18027774e+0
6,
        2.06562929e+06,  2.51380398e+04,  5.78865412e+03,  9.16700739e+0
5,
        5.78708232e+06, -2.77153109e+06,  2.56506093e+06,  7.96799681e+0
2,
        1.66940590e+06, -7.83122495e+06,  4.11555049e+07,  1.66582573e+0
7,
        4.67766172e+05, -2.10039839e+07, -6.05854738e+06,  4.56611429e+0
7,
        1.05890668e+07,  2.36241181e+07, -2.59876748e+07, -6.83068831e+0
7,
        3.45968177e+06, -7.81046137e+06,  1.58122451e+07,  1.15163012e+0
7,
        1.23596634e+07, -1.81783723e+05,  1.77935128e+03, -1.12652023e+0
7,
        1.20685944e+07,  1.84482107e+06, -5.15209274e+05,  2.32970156e+0
4,
       -1.83874561e+06,  2.33338407e+03, -4.18658204e+03, -8.73650881e+0
7,
       -1.70062492e+07,  9.97911063e+05, -2.11605581e+06, -5.28445122e+0
6,
```

```
        -4.11582569e+06,  7.69656832e+06, -6.45125577e+06,  2.33474373e+0
7,
        -1.30757965e+06,  9.21030439e+06, -3.80571113e+06, -5.47114659e+0
6,
        -1.21392493e+06,  2.74415678e+07, -1.04272988e+06, -2.37431314e+0
5,
        -9.45871876e+06, -1.84278340e+07, -8.49964821e+06, -2.44489302e+0
7,
         2.10758801e+06,  4.71471985e+06, -1.12469003e+08, -5.18224881e+0
3,
        -1.26334341e+07, -1.95247503e+07,  2.80692671e+07, -2.77822386e+0
6,
        -9.99947201e+05,  2.51770774e+07,  2.38375385e+06, -5.41639135e+0
4,
         7.17122597e+03,  1.13375665e+07,  8.67758109e+05,  7.61394230e+0
5,
        -1.02020751e+08,  5.61871341e+05,  4.50467921e+06, -1.84903095e+0
3,
         9.08249594e+06,  3.53724044e+05,  5.13092176e+07, -2.44612920e+0
6,
        -1.57531879e+06, -8.69634578e+06,  4.37882430e+06,  7.08257765e+0
4,
         2.55478519e+07, -2.18249009e+06, -1.03542687e+07,  1.32870721e+0
7,
         1.42160485e+06, -3.69344870e+06,  1.83210730e+06, -7.42271810e+0
4,
         1.34361553e+07,  8.27736075e+06,  2.40289995e+05,  7.36665422e+0
5,
        -1.82834395e+06, -1.50676578e+04, -4.47210219e+06, -1.71369048e+0
6,
        -6.57857741e+05, -5.82407128e+06,  2.02324276e+04,  1.20871051e+0
7,
         2.73900974e+06,  3.23509105e+03,  2.27735949e+07, -5.03022585e+0
4,
        -7.59591779e+05, -9.66263231e+06,  1.23865800e+07,  1.18355472e+0
5,
        -3.27351009e+07,  1.04322461e+07,  6.50814358e+05, -4.87315136e+0
6,
        -1.06178708e+07, -2.63294787e+07,  6.55733475e+07, -3.89085879e+0
4,
         2.88686226e+06,  1.65125450e+07,  6.47511789e+06, -2.16998199e+0
7,
        -1.81797076e+06, -1.98523740e+07, -8.03188499e+05,  1.38048024e+0
6,
        -2.17362458e+06,  8.45532477e+06,  4.62260169e+06, -9.75474690e+0
6,
        -1.69340277e+05, -1.35005641e+06,  4.64444106e+06,  7.31504101e+0
6,
        -4.42963666e+06,  7.03759009e+06, -7.27380821e+07,  2.96415970e+0
5,
        -2.53204814e+07, -2.06370690e+02,  5.66948322e+06,  1.08963863e+0
3,
        -5.34876426e+06, -2.12026701e+07,  1.74648615e+06,  7.55190848e+0
6,
         2.59735087e+07,  4.90624316e+06,  2.90641505e+07,  1.10533728e+0
6,
         2.52774388e+07,  3.68959840e+06,  8.71378846e+05,  3.10859910e+0
```

```
7,
        -7.27977462e+05,  2.82809880e+06, -7.02817966e+04,  2.98179959e+0
6,
        -3.75454208e+06,  4.73797687e+06,  1.77631342e+06, -2.06167729e+0
5,
         1.57806591e+07,  7.73328265e+06, -2.44544682e+04,  2.15521669e+0
7,
         5.90723400e+06,  3.60403024e+07,  3.37041097e+06,  1.34451294e+0
6,
        -8.06068182e+06,  3.12905063e+07,  4.51484735e+06, -1.34546899e+0
7,
        -1.64714454e+06,  1.37891486e+07,  1.23268253e+05,  1.20176316e+0
7,
         4.17374316e+04,  1.66890996e+06, -1.48254718e+06,  1.13750076e+0
6,
         1.20139056e+07,  4.17066474e+05, -2.20309167e+07, -1.19878290e+0
3,
        -1.61335701e+04, -3.48357769e+05, -2.58305776e+07,  2.08858554e+0
7,
         9.72219274e+06, -5.34714998e+05,  9.61766217e+05,  5.05237804e+0
6,
        -4.80281932e+07,  5.31881194e+04, -4.38627554e+06,  7.62988289e+0
6,
        -9.94952246e+05, -2.54221877e+06,  7.26450307e+05,  5.47976652e+0
6,
        -8.20663117e+06, -1.77801086e+03, -2.34828795e+02,  2.29816193e+0
2,
        -1.24938635e+03, -3.03975120e+03])
```

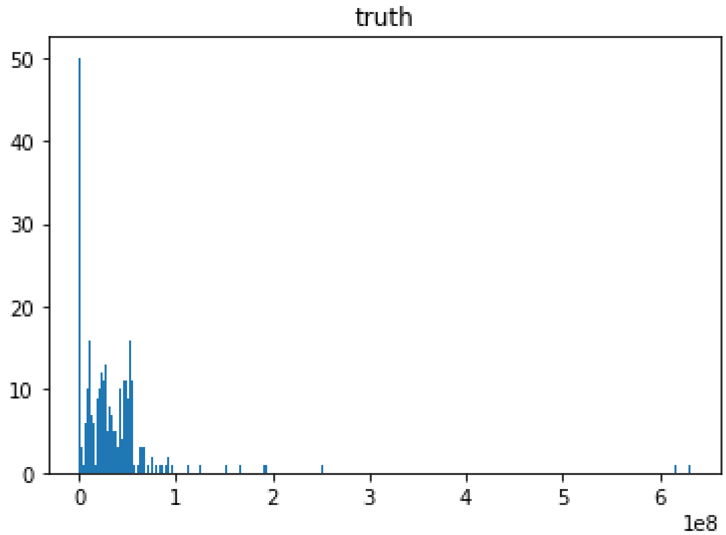
In [39]:
```
plt.hist(reaggregated_predictions, bins = 300)
plt.title('predictions')
```

Out[39]:
```
Text(0.5, 1.0, 'predictions')
```



In [40]:
```
plt.hist(reaggregated_truths, bins = 300)
plt.title('truth')
```

Out[40]: Text(0.5, 1.0, 'truth')

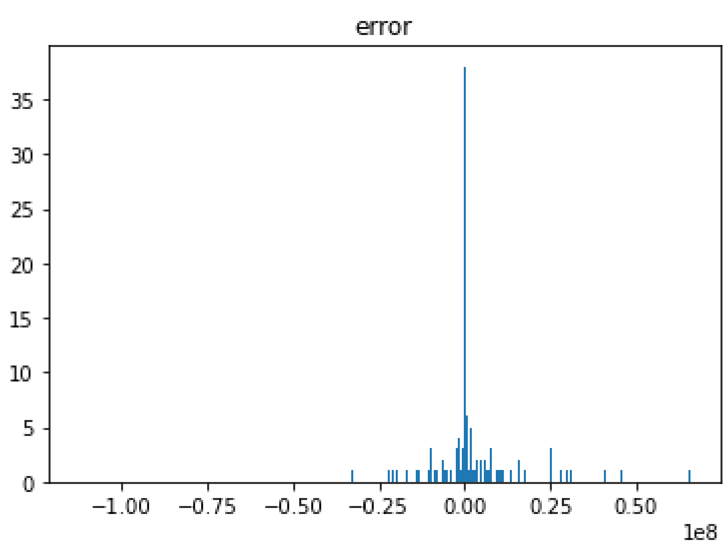


In [41]:
```
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error( test_labels, predicted_labels )
```

Out[41]: 0.2021484466892188

In [42]:
```
plt.hist(e, bins = 1000)
plt.title('error')
```

Out[42]: Text(0.5, 1.0, 'error')

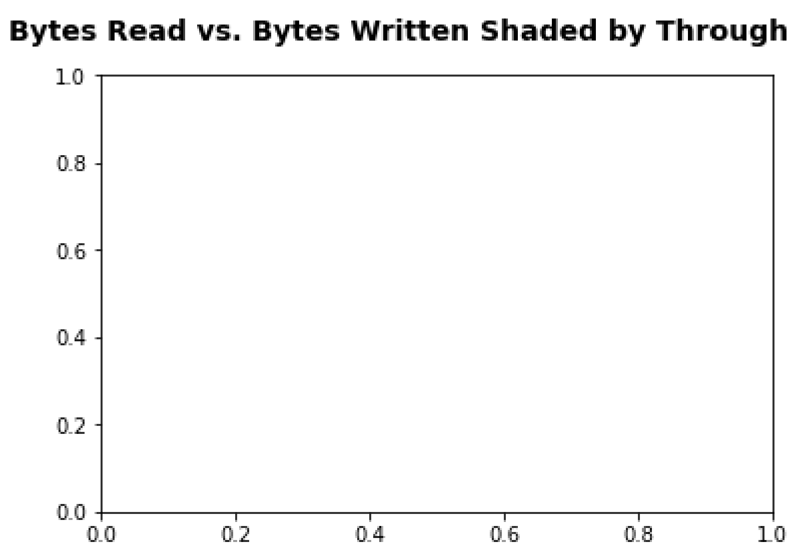


In [ ]:

In [43]:

```
fig = plt.figure()
fig.suptitle('Bytes Read vs. Bytes Written Shaded by Throughput', fontsize=

ax = fig.add_subplot(111)
sp = ax.scatter(f0['posix_bytes_read'],d['posix_bytes_write'], marker = 'x'

ax.set_xlabel('Bytes Read s')
ax.set_ylabel('Bytes Written s')
ax.loglog()
#plt.autoscale(enable=True, axis='y')
plt.xlim(10**0,10**15)
plt.ylim(10**0,10**15)
fig.colorbar(sp)
plt.show()
```

```
---------------------------------------------------------------------------
--
IndexError                                Traceback (most recent call las
t)
<ipython-input-43-1a3c85cc9c15> in <module>
      3
      4 ax = fig.add_subplot(111)
----> 5 sp = ax.scatter(f0['posix_bytes_read'],d['posix_bytes_write'], ma
rker = 'x',c = np.log10(t['throughput']),cmap='viridis')
      6
      7 ax.set_xlabel('Bytes Read s')

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis
 (`None`) and integer or boolean arrays are valid indices
```



In [ ]:

In [ ]: