ChronicleDB on a Raft: Performance Tests

In [1]:
```
import pandas as pd
import numpy as np
import altair as alt

#pip install vega
#jupyter nbextension install --sys-prefix --py vega

#alt.renderers.enable('notebook')
alt.renderers.enable('html')
```

Out[1]:
```
RendererRegistry.enable('html')
```

In [58]:
```
import requests

#host = 'http://localhost:8080/api'
host = 'http://3.121.183.166:8080/api'


def run_test(count, target='event-store'):
    query = { "batchSize": count }
    response = requests.request('GET', f'{host}/sys-info/performance/measure/{target}/insert-events/{count}', params=
    return response.json()

def clear_stream(stream='demo-event-store', target='event-store'):
    response = requests.request('DELETE', f'{host}/{target}/streams/{stream}/events')
    token = response.json()
    print(token)
    response = requests.request('POST', f'{host}/{target}/clear-request-confirmation', json=token)
    return response
```

In [59]:
```
clear_stream('demo_event_store', target='event-store/embedded')
```

Out[59]:
```
{'streamName': 'demo_event_store', 'token': '8_1gIecvKQimY9cpEha522qhH0UwDYSn'}
<Response [200]>
```

In [60]:
```
# TODO test on non-replicated chronicle db and compare

# To create streams in the embedded db
def create_schema_for_embedded_db(schema):
    response = requests.request('POST', f'{host}/event-store/embedded/streams', json=schema)
    return response
```

In [61]:
```
create_schema_for_embedded_db({
    "streamName": "demo_event_store",
    "schema": [
        {
            "name": "SYMBOL",
            "type": "STRING",
            "properties": {}
        },
        {
            "name": "SECURITYTYPE",
            "type": "INTEGER",
            "properties": {}
        },
        {
            "name": "LASTTRADEPRICE",
            "type": "FLOAT",
            "properties": {}
        }
    ]
})
```

Out[61]:
```
<Response [500]>
```

In [62]:
```
import datetime
import pandas as pd

def run_tests(counts=[1,100], trials_per_count=1, env_info={}, target='event-store', stream='demo-event-store', csv_c
    now = datetime.datetime.now().replace(microsecond=0).isoformat().replace(':', '-')
    csv_name = f'results-{now}.csv'

    env_info_columns = list(env_info.keys())

    df = pd.DataFrame(columns = [*env_info_columns, 'buffer_size_in_bytes', 'event_count', 'trial', 'duration_in_ms',

    for count in counts:
        for i in range(0, trials_per_count):
            print(f'Trial {i+1} for event count {count}')
```

```
            results = run_test(count, target)
            print(results['message'])
            df = df.append({
                **env_info,
                'buffer_size_in_bytes': results['bufferSize'],
                'trial': i+1,
                'event_count': count,
                'duration_in_ms': results['timeElapsed'],
                'measured_on': now
            }, ignore_index=True)
            df.to_csv(f'measurements/{csv_dir}/{csv_name}', index=False)
            # clean service after each trial. Must delete all events to avoid OOO
            stream_cleared = clear_stream(stream, target)
            print(stream_cleared)

    return df
```

In [14]:
```
clear_stream()
```

```
{'streamName': 'demo-event-store', 'token': 'wBINNQQtadtotOFZKgo-jathuDzaJlNc'}
<Response [200]>
```

Out[14]:

In [72]:
```
env_info = {
    #'cluster_type': 'localMacBookProIntelI9',
    'cluster_type': 'awsLightsail2GB',
    'node_count': 3,
    'event_type': 'randomized',
    'buffer_type': 'blocking'
}

run_tests(counts=[1, 100, 10000, 1000000], trials_per_count=10, env_info=env_info, target='event-store', csv_dir='aws

# run_tests(counts=[10000000], trials_per_count=10, env_info=env_info)

# an event of our examples has ~21 bytes
```

In [64]:
```
# measure performance of non-replicated, embedded store (the original one)

env_info = {
    'cluster_type': 'standalone-embedded-aws',
    'node_count': 1,
    'event_type': 'randomized',
    'buffer_type': 'none-embedded'
}

run_tests(counts=[1, 100, 10000, 1000000], trials_per_count=10, env_info=env_info, target='event-store/embedded', str
```

Out[64]:

	cluster_type	node_count	event_type	buffer_type	buffer_size_in_bytes	event_count	trial	duration_in_ms	measured_on
0	standalone-embedded-aws	1	randomized	none-embedded	0	1	1	6	2022-03-16T15-44-04
1	standalone-embedded-aws	1	randomized	none-embedded	0	1	2	6	2022-03-16T15-44-04
2	standalone-embedded-aws	1	randomized	none-embedded	0	1	3	147	2022-03-16T15-44-04
3	standalone-embedded-aws	1	randomized	none-embedded	0	1	4	9	2022-03-16T15-44-04
4	standalone-embedded-aws	1	randomized	none-embedded	0	1	5	4	2022-03-16T15-44-04
5	standalone-embedded-aws	1	randomized	none-embedded	0	1	6	12	2022-03-16T15-44-04
6	standalone-embedded-aws	1	randomized	none-embedded	0	1	7	7	2022-03-16T15-44-04
7	standalone-embedded-aws	1	randomized	none-embedded	0	1	8	9	2022-03-16T15-44-04
8	standalone-embedded-aws	1	randomized	none-embedded	0	1	9	2	2022-03-16T15-44-04
9	standalone-embedded-aws	1	randomized	none-embedded	0	1	10	0	2022-03-16T15-44-04
10	standalone-embedded-aws	1	randomized	none-embedded	0	100	1	86	2022-03-16T15-44-04
11	standalone-embedded-aws	1	randomized	none-embedded	0	100	2	3	2022-03-16T15-44-04
12	standalone-embedded-aws	1	randomized	none-embedded	0	100	3	3	2022-03-16T15-44-04
13	standalone-embedded-aws	1	randomized	none-embedded	0	100	4	22	2022-03-16T15-44-04
14	standalone-embedded-aws	1	randomized	none-embedded	0	100	5	15	2022-03-16T15-44-04
15	standalone-embedded-aws	1	randomized	none-embedded	0	100	6	3	2022-03-16T15-44-04
16	standalone-embedded-aws	1	randomized	none-embedded	0	100	7	3	2022-03-16T15-44-04
17	standalone-embedded-aws	1	randomized	none-embedded	0	100	8	9	2022-03-16T15-44-04
18	standalone-embedded-aws	1	randomized	none-embedded	0	100	9	6	2022-03-16T15-44-04
19	standalone-embedded-aws	1	randomized	none-embedded	0	100	10	26	2022-03-16T15-44-04
20	standalone-embedded-aws	1	randomized	none-embedded	0	10000	1	60	2022-03-16T15-44-04

	cluster_type	node_count	event_type	buffer_type	buffer_size_in_bytes	event_count	trial	duration_in_ms	measured_on
21	standalone-embedded-aws	1	randomized	none-embedded	0	10000	2	34	2022-03-16T15-44-04
22	standalone-embedded-aws	1	randomized	none-embedded	0	10000	3	23	2022-03-16T15-44-04
23	standalone-embedded-aws	1	randomized	none-embedded	0	10000	4	28	2022-03-16T15-44-04
24	standalone-embedded-aws	1	randomized	none-embedded	0	10000	5	9	2022-03-16T15-44-04
25	standalone-embedded-aws	1	randomized	none-embedded	0	10000	6	15	2022-03-16T15-44-04
26	standalone-embedded-aws	1	randomized	none-embedded	0	10000	7	19	2022-03-16T15-44-04
27	standalone-embedded-aws	1	randomized	none-embedded	0	10000	8	141	2022-03-16T15-44-04
28	standalone-embedded-aws	1	randomized	none-embedded	0	10000	9	15	2022-03-16T15-44-04
29	standalone-embedded-aws	1	randomized	none-embedded	0	10000	10	19	2022-03-16T15-44-04
30	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	1	580	2022-03-16T15-44-04
31	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	2	422	2022-03-16T15-44-04
32	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	3	554	2022-03-16T15-44-04
33	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	4	460	2022-03-16T15-44-04
34	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	5	440	2022-03-16T15-44-04
35	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	6	398	2022-03-16T15-44-04
36	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	7	507	2022-03-16T15-44-04
37	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	8	388	2022-03-16T15-44-04
38	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	9	440	2022-03-16T15-44-04
39	standalone-embedded-aws	1	randomized	none-embedded	0	1000000	10	582	2022-03-16T15-44-04

In [23]:
```
# TODO we actually don't know how long the insert is in total as we are async as soon as events are offered to the bu
# So, as we are waiting for buffer flushes, but not the last flush, we miss the time it tooks to apply the last flush
# TODO therefore need some callback in event store to know when the store is done, not just the buffer...
# example: Inserted events 10000 times asynchronously [45ms] <-- this is just the time it takes to insert them into t
# -> at least 4.5 sec for 1mio events to put into the buffer

# -> raft log entry size is limited! ->  # Log entry size 13383315 exceeds the max buffer limit of 4194304
# raft.server.log.appender.buffer.byte-limit = 4MB
```

In [ ]:
```
# TODO test with maximum size buffer
```

In [ ]:
```
# TODO test with raft log entries not persisted
```

Evaluating the benchmark results

All of this is currently under conditions without out-of-order events

In [2]:
```
# load all measurements

from os import walk

resource_folder = 'measurements'

resources = next(walk(resource_folder), (None, None, []))[2]

print(f'{len(resources)} files in total')
```

```
24 files in total
```

In [3]:
```
datasets = {filename: pd.read_csv(f'{resource_folder}/{filename}', sep=',', encoding='utf-8', error_bad_lines=False)
```

```
/var/folders/mb/g5kpvlh145j5qpm9s7fmb08c0000gn/T/ipykernel_85187/3005144025.py:1: FutureWarning: The error_bad_lines
argument has been deprecated and will be removed in a future version.


  datasets = {filename: pd.read_csv(f'{resource_folder}/{filename}', sep=',', encoding='utf-8', error_bad_lines=Fals
e) for filename in resources if not filename.startswith(".")}
```

In [4]:
```
all_measurements_df = pd.concat(datasets, ignore_index=True)

all_measurements_df
```

Out[4]:

	cluster_type	node_count	event_type	buffer_type	buffer_size_in_bytes	event_count	trial	duration_in_ms	measured_on
0	local17	3	randomized	blocking	1048576	10000000	1	124276	2022-01-29T15-16-22
1	local17	3	randomized	blocking	1048576	10000000	2	122494	2022-01-29T15-16-22
2	local17	3	randomized	blocking	1048576	10000000	3	117724	2022-01-29T15-16-22

	cluster_type	node_count	event_type	buffer_type	buffer_size_in_bytes	event_count	trial	duration_in_ms	measured_on
3	localI7	3	randomized	blocking	1048576	10000000	4	119596	2022-01-29T15-16-22
4	localI7	3	randomized	blocking	1048576	10000000	5	114629	2022-01-29T15-16-22
...	...	...	...	...	...	...	...	...	...
830	localI7	3	randomized	blocking	10240	1000000	6	20389	2022-01-29T12-48-43
831	localI7	3	randomized	blocking	10240	1000000	7	20635	2022-01-29T12-48-43
832	localI7	3	randomized	blocking	10240	1000000	8	21030	2022-01-29T12-48-43
833	localI7	3	randomized	blocking	10240	1000000	9	21108	2022-01-29T12-48-43
834	localI7	3	randomized	blocking	10240	1000000	10	21375	2022-01-29T12-48-43

835 rows × 9 columns

In [5]:

```
# supress warnings
pd.options.mode.chained_assignment = None

# remove rows with duration = 0 to avoid divide by zero
all_measurements_df = all_measurements_df[all_measurements_df['duration_in_ms'] > 0]

# replace buffer_size 0 with 1 to allow log scale plotting
all_measurements_df.loc[all_measurements_df['buffer_size_in_bytes'] == 0, 'buffer_size_in_bytes'] = 1

all_measurements_df['duration_per_event'] = all_measurements_df['duration_in_ms'] / all_measurements_df['event_count']
all_measurements_df['duration_per_mio_events'] = all_measurements_df['duration_per_event'] * 1000000
all_measurements_df['duration_per_mio_events_in_sec'] = all_measurements_df['duration_per_mio_events'] / 1000
all_measurements_df['events_per_second'] = 1000 / all_measurements_df['duration_per_event']

pd.options.mode.chained_assignment = 'warn'

all_measurements_df.describe()
```

Out[5]:

	node_count	buffer_size_in_bytes	event_count	trial	duration_in_ms	duration_per_event	duration_per_mio_events	duration_per_mio_eve
count	743.000000	7.430000e+02	7.430000e+02	743.000000	743.000000	743.000000	7.430000e+02	7
mean	3.189771	6.178415e+05	3.998901e+05	5.425303	14820.282638	2.483687	2.483687e+06	24
std	1.155231	5.059331e+05	1.204331e+06	2.891013	61681.256728	12.500844	1.250084e+07	125
min	1.000000	1.000000e+00	1.000000e+00	1.000000	1.000000	0.000388	3.880000e+02	
25%	3.000000	1.024000e+04	1.000000e+02	3.000000	4.000000	0.008908	8.908000e+03	
50%	3.000000	1.048576e+06	1.000000e+04	5.000000	37.000000	0.020000	2.000000e+04	
75%	3.000000	1.048576e+06	1.000000e+06	8.000000	7308.000000	0.225000	2.250000e+05	2
max	6.000000	1.048576e+06	1.000000e+07	10.000000	485271.000000	246.000000	2.460000e+08	2460

In [6]:

```
#replicated_measurements_df = all_measurements_df[all_measurements_df['cluster_type'] != 'standalone-embedded']
replicated_measurements_df = all_measurements_df[all_measurements_df['cluster_type'] == 'localI7']
replicated_measurements_3_nodes_df = replicated_measurements_df[replicated_measurements_df['node_count'] == 3]
embedded_measurements_df = all_measurements_df[all_measurements_df['cluster_type'] == 'standalone-embedded']
remote_replicated_measurements_df = all_measurements_df[all_measurements_df['cluster_type'] == 'awsLightsail2GB']

local_and_remote_replicated_measurements_3_nodes_df = all_measurements_df[all_measurements_df['node_count'] == 3]
```

In [7]:

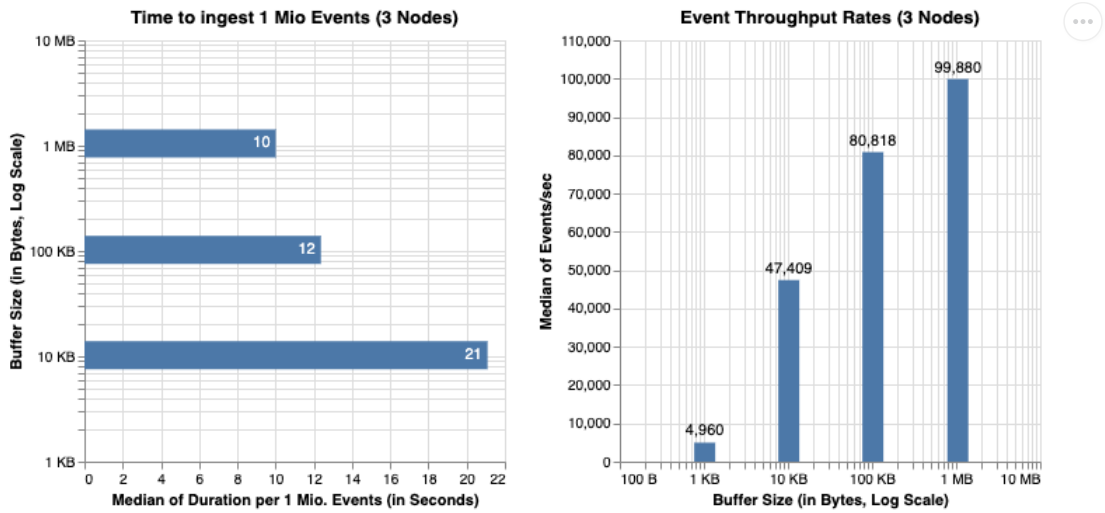
```
import altair as alt

def plot_event_rate(df, title="Event Throughput Rates", show_benchmark=False, show_benchmark_at=10000000, aggregate='
    buffer_size_label_expr = "datum.label && datum.label[0] == '1' ? (datum.value >= 1000000 ? datum.value / 1000000
    if show_benchmark:
        buffer_size_label_expr = f'datum.value == {show_benchmark_at} ? "Standalone" : ({buffer_size_label_expr})'

    blocking_queue_event_rate_plot = alt.Chart(df).mark_bar(clip=True, width=15).encode(
        x=alt.X('buffer_size_in_bytes:Q', scale=alt.Scale(type='log'), title="Buffer Size (in Bytes, Log Scale)", axi
            labelExpr=buffer_size_label_expr
        )),
        y=alt.Y(f'{aggregate}(events_per_second):Q', title=f"{aggregate.capitalize()} of Events/sec", scale=alt.Scale
        color=alt.condition(
            alt.datum.buffer_size_in_bytes == show_benchmark_at,
            alt.value('#919eac'),
            alt.value('#4c78a8')
        )
    ).properties(width=300, height=300, title=title)

    return blocking_queue_event_rate_plot + blocking_queue_event_rate_plot.mark_text(
        align='center',
        color='black',
        dx=0,
        dy=-8
    ).encode(
        text=alt.Text(f'{aggregate}(events_per_second):Q', format=',.0f'),
```

```
        color=alt.value('black')
    ).transform_calculate(label='datum.y + " inches"')
```

In [8]:

```
buffer_size_label_expr = "datum.label && datum.label[0] == '1' ? (datum.value >= 1000000 ? datum.value / 1000000 + '

duration_plot = alt.Chart(replicated_measurements_3_nodes_df[replicated_measurements_3_nodes_df['buffer_size_in_bytes
    x=alt.X('median(duration_per_mio_events_in_sec):Q', title="Median of Duration per 1 Mio. Events (in Seconds)"),
    y=alt.Y('buffer_size_in_bytes:Q', scale=alt.Scale(type='log'), title="Buffer Size (in Bytes, Log Scale)", axis=al
        labelExpr=buffer_size_label_expr
    )),
).properties(width=300, title="Time to ingest 1 Mio Events (3 Nodes)")

duration_plot = duration_plot + duration_plot.mark_text(
    align='right',
    color='white',
    dx=-4,
    #dy=-18
).encode(
    text=alt.Text('median(duration_per_mio_events_in_sec):Q', format=',.0f'),
)

duration_plot | plot_event_rate(replicated_measurements_3_nodes_df[replicated_measurements_3_nodes_df['buffer_size_in
```

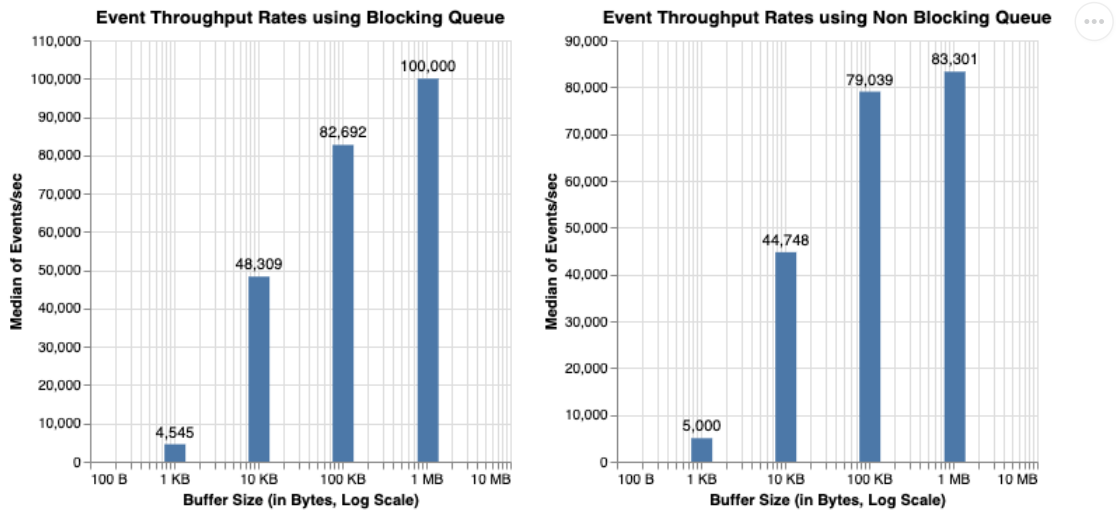
Out[8]:



In [9]:

```
blocking_queue_df = replicated_measurements_3_nodes_df[replicated_measurements_3_nodes_df['buffer_type'] == 'blocking
non_blocking_queue_df = replicated_measurements_3_nodes_df[replicated_measurements_3_nodes_df['buffer_type'] == 'non-

plot_event_rate(blocking_queue_df, title="Event Throughput Rates using Blocking Queue") | plot_event_rate(non_blockin
```

Out[9]:

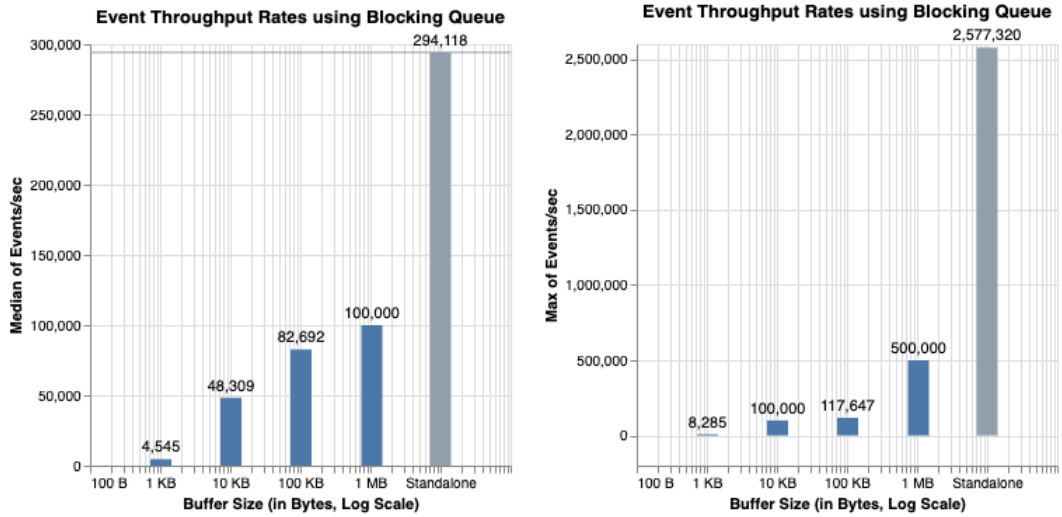


In [22]:

```
blocking_queue_vs_standalone_df = all_measurements_df[((all_measurements_df['buffer_type'] == 'blocking') & (all_meas

show_benchmark_at = 10000000

blocking_queue_vs_standalone_df.loc[(blocking_queue_vs_standalone_df['buffer_type'] == 'none-embedded'), 'buffer_size

median_plot = plot_event_rate(blocking_queue_vs_standalone_df, title="Event Throughput Rates using Blocking Queue", s
max_plot = plot_event_rate(blocking_queue_vs_standalone_df, title="Event Throughput Rates using Blocking Queue", show

benchmark_score = all_measurements_df[all_measurements_df['buffer_type'] == 'none-embedded']['events_per_second'].mea

median_plot + alt.Chart(pd.DataFrame({'y': [benchmark_score]})).mark_rule(color="#919eac").encode(y='y') | max_plot
```

```
# a = plot_event_rate(all_measurements_df[(all_measurements_df['buffer_type'] == 'blocking')], title="Event Throughpu
# b = plot_event_rate(all_measurements_df[(all_measurements_df['buffer_type'] == 'none-embedded')], title="Event Thro

# a + b
```

Out[22]:



Comparison of Local Cluster (Single Machine) vs. Remote Cluster on AWS

In [11]:

```
# TODO render throughput for blocking queue with 1MB buffer on local 3 nodes

local_vs_remote_df = local_and_remote_replicated_measurements_3_nodes_df[
    (local_and_remote_replicated_measurements_3_nodes_df['buffer_size_in_bytes'] == 1048576)
    & (local_and_remote_replicated_measurements_3_nodes_df['buffer_type'] == 'blocking')]



# TODO render throughput for blocking queue with 1MB buffer on aws

# TODO maybe later integrate in plots above
```

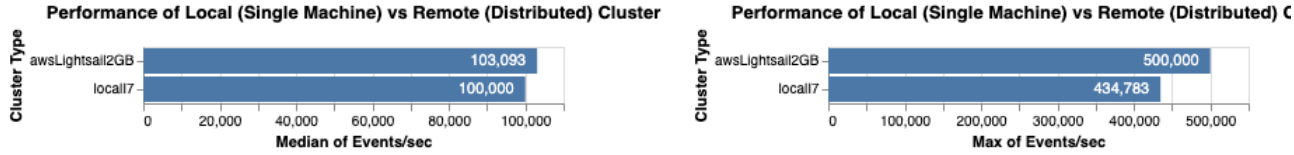
In [12]:

```
def compare_local_vs_remote_plot(aggregate='median', max_scale=550000):
    plot = alt.Chart(local_vs_remote_df).mark_bar(clip=True).encode(
        x=alt.X(f'{aggregate}(events_per_second):Q', title=f'{aggregate.capitalize()} of Events/sec', scale=alt.Scale
        y=alt.Y('cluster_type:N', title="Cluster Type"),
    ).properties(width=300, title="Performance of Local (Single Machine) vs Remote (Distributed) Cluster")

    plot = plot + plot.mark_text(
        align='right',
        color='white',
        dx=-8,
        dy=0
    ).encode(
        text=alt.Text(f'{aggregate}(events_per_second):Q', format=',.0f'),
    )

    return plot
```

In [13]:

```
compare_local_vs_remote_plot(aggregate='median', max_scale=110000) | compare_local_vs_remote_plot(aggregate='max', ma
```

Out[13]:



- [ ] multiple nodes on AWS
- [ ] benchmarking against standalone on aws node

Comparison of Different Cluster Sizes (# of Nodes)

Evaluation has been run on local machine

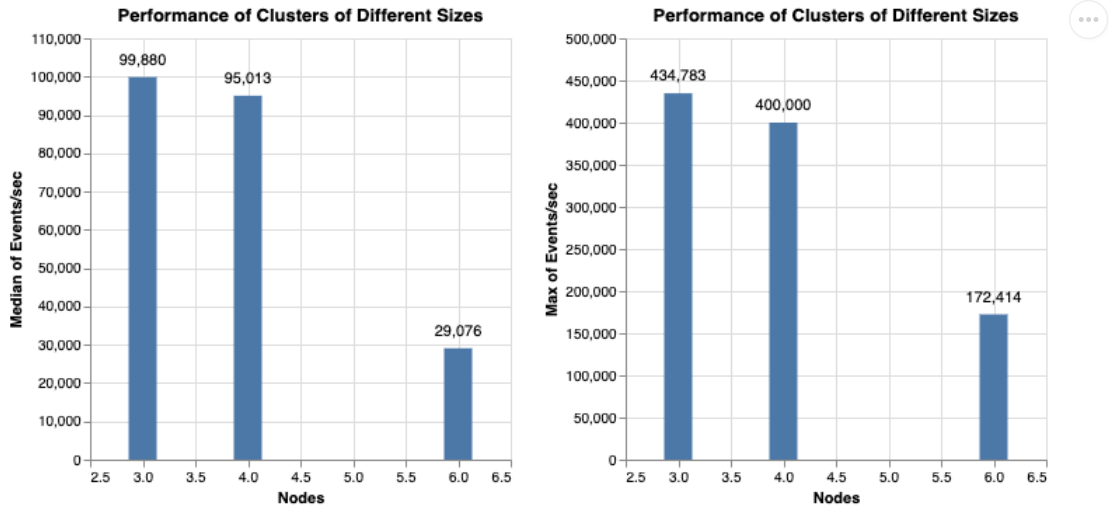
In [14]:

```
def compare_nodes_plot(aggregate='median', max_scale=550000):
    node_compare_plot = alt.Chart(replicated_measurements_df[replicated_measurements_df['buffer_size_in_bytes'] == 10
        y=alt.Y(f'{aggregate}(events_per_second):Q', title=f'{aggregate.capitalize()} of Events/sec', scale=alt.Scale
        x=alt.X('node_count:Q', title="Nodes"),
```

```
    ).properties(width=300, title="Performance of Clusters of Different Sizes")

    node_compare_plot = node_compare_plot + node_compare_plot.mark_text(
        align='center',
        color='black',
        dx=0,
        dy=-12
    ).encode(
        text=alt.Text(f'{aggregate}(events_per_second):Q', format=',.0f'),
    )

    return node_compare_plot
```

In [15]:
```
compare_nodes_plot(aggregate='median', max_scale=110000) | compare_nodes_plot(aggregate='max', max_scale=500000)
```

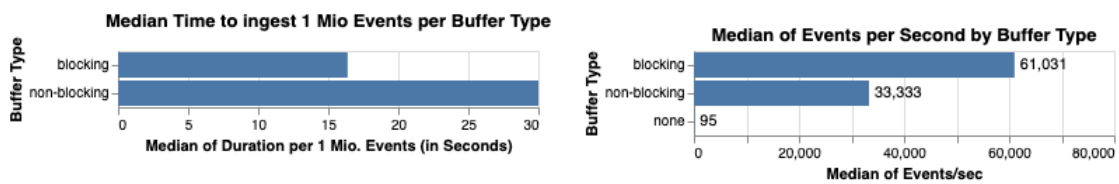
Out[15]:



Comparison of Blocking vs Non-Blocking Buffer

In [16]:
```
buffered_measurements_df = replicated_measurements_df[replicated_measurements_df['buffer_type'] != 'none']

time_per_buffer_plot = alt.Chart(buffered_measurements_df).mark_bar(clip=True).encode(
    x=alt.X('median(duration_per_mio_events_in_sec):Q', title="Median of Duration per 1 Mio. Events (in Seconds)"),
    y=alt.Y('buffer_type:N',title="Buffer Type"),
).properties(width=300, title="Median Time to ingest 1 Mio Events per Buffer Type")

events_per_sec_plot = alt.Chart(replicated_measurements_df).mark_bar(clip=True).encode(
    x=alt.X('median(events_per_second):Q', title="Median of Events/sec", scale=alt.Scale(domain=[0, 80000])),
    y=alt.Y('buffer_type:N',title="Buffer Type"),
).properties(width=300, title="Median of Events per Second by Buffer Type")


events_per_sec_plot = events_per_sec_plot + events_per_sec_plot.mark_text(
    align='left',
    color='black',
    dx=4,
    #dy=-18
).encode(
    text=alt.Text('median(events_per_second):Q', format=',.0f'),
)

time_per_buffer_plot | events_per_sec_plot
```

Out[16]:

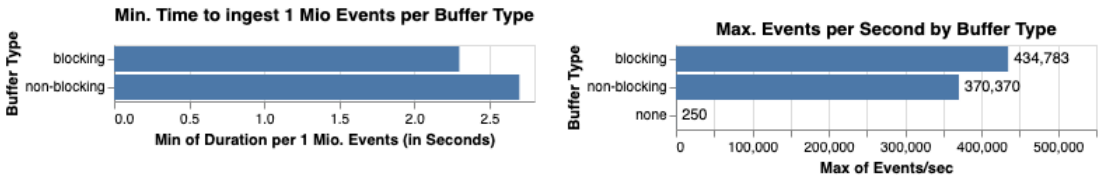


Comparison with Standalone/Embedded ChronicleDB

In [17]:
```
time_per_buffer_plot = alt.Chart(buffered_measurements_df).mark_bar(clip=True).encode(
    x=alt.X('min(duration_per_mio_events_in_sec):Q', title="Min of Duration per 1 Mio. Events (in Seconds)"),
    y=alt.Y('buffer_type:N',title="Buffer Type"),
).properties(width=300, title="Min. Time to ingest 1 Mio Events per Buffer Type")

events_per_sec_plot = alt.Chart(replicated_measurements_df).mark_bar(clip=True).encode(
    x=alt.X('max(events_per_second):Q', title="Max of Events/sec", scale=alt.Scale(domain=[0, 550000])),
    y=alt.Y('buffer_type:N',title="Buffer Type"),
).properties(width=300, title="Max. Events per Second by Buffer Type")
```

```
events_per_sec_plot = events_per_sec_plot + events_per_sec_plot.mark_text(
    align='left',
    color='black',
    dx=4,
    #dy=-18
).encode(
    text=alt.Text('max(events_per_second):Q', format=',.0f'),
)

time_per_buffer_plot | events_per_sec_plot
```

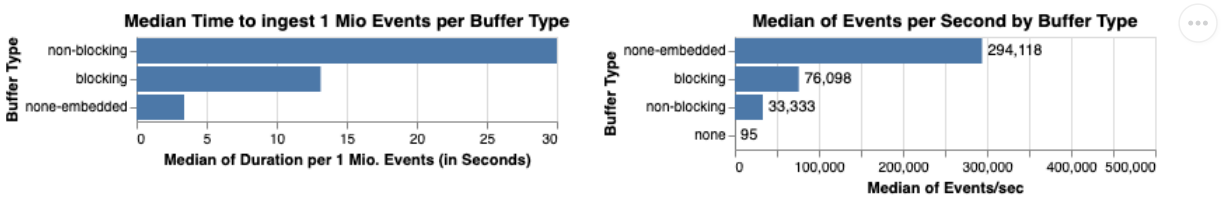
Out[17]:



In [18]:

```
buffered_and_embedded_measurements_df = all_measurements_df[all_measurements_df['buffer_type'] != 'none']

time_per_buffer_plot = alt.Chart(buffered_and_embedded_measurements_df).mark_bar(clip=True).encode(
    x=alt.X('median(duration_per_mio_events_in_sec):Q', title="Median of Duration per 1 Mio. Events (in Seconds)"),
    y=alt.Y('buffer_type:N',title="Buffer Type", sort='-x'),
).properties(width=300, title="Median Time to ingest 1 Mio Events per Buffer Type")

events_per_sec_plot = alt.Chart(all_measurements_df).mark_bar(clip=True).encode(
    x=alt.X('median(events_per_second):Q', title="Median of Events/sec", scale=alt.Scale(domain=[0, 500000])),
    y=alt.Y('buffer_type:N',title="Buffer Type", sort='-x'),
).properties(width=300, title="Median of Events per Second by Buffer Type")


events_per_sec_plot = events_per_sec_plot + events_per_sec_plot.mark_text(
    align='left',
    color='black',
    dx=4,
    #dy=-18
).encode(
    text=alt.Text('median(events_per_second):Q', format=',.0f'),
)

time_per_buffer_plot | events_per_sec_plot
```

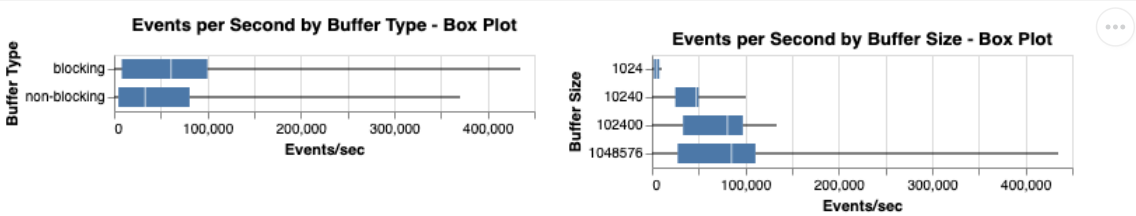
Out[18]:



In [19]:

```
alt.Chart(buffered_measurements_df).mark_boxplot(extent='min-max').encode(
    y=alt.Y('buffer_type:N',title="Buffer Type"),
    x=alt.X('events_per_second:Q', title="Events/sec"),
).properties(width=300, title="Events per Second by Buffer Type - Box Plot") | alt.Chart(buffered_measurements_df).ma
    y=alt.Y('buffer_size_in_bytes:O',title="Buffer Size"),
    x=alt.X('events_per_second:Q', title="Events/sec"),
).properties(width=300, title="Events per Second by Buffer Size - Box Plot")
```

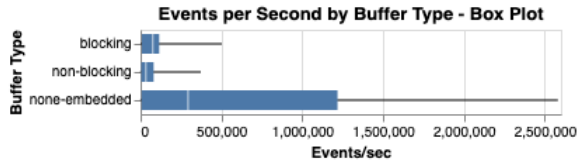
Out[19]:

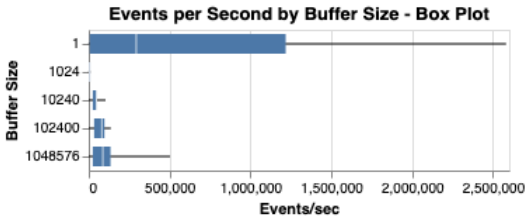


In [20]:

```
alt.Chart(buffered_and_embedded_measurements_df).mark_boxplot(extent='min-max').encode(
    y=alt.Y('buffer_type:N',title="Buffer Type"),
    x=alt.X('events_per_second:Q', title="Events/sec"),
).properties(width=300, title="Events per Second by Buffer Type - Box Plot") | alt.Chart(buffered_and_embedded_measur
    y=alt.Y('buffer_size_in_bytes:O',title="Buffer Size"),
    x=alt.X('events_per_second:Q', title="Events/sec"),
).properties(width=300, title="Events per Second by Buffer Size - Box Plot")
```

Out[20]:

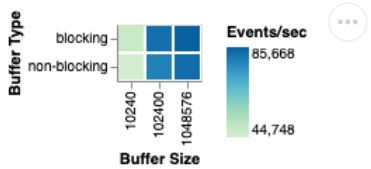




In [21]:

```
alt.Chart(replicated_measurements_df[replicated_measurements_df['buffer_size_in_bytes'] > 1024]).mark_bar(clip=True).
    color=alt.Color('median(events_per_second):Q', title="Events/sec", scale=alt.Scale(scheme='greenblue')),
    x=alt.X('buffer_size_in_bytes:N',title="Buffer Size"),
    y=alt.Y('buffer_type:N',title="Buffer Type"),
).properties(title="")
```

Out[21]:



Running the cluster without a buffer leads to 100% utilization of the machines IO, as in the current naive implementation of the raft log and state machine, each event is sent to all nodes, needs to be committed by at least a quorum of notes and is also written into the raft log of each node.

```
{localHostName=ip-172-26-0-78.eu-central-1.compute.internal, javaVersion=11,
localHostAddress=172.26.0.78, jdkVersion=11.0.14.1, storagePath=/home/ec2-user/chronicledb,
osName=Linux, springVersion=5.3.9, osVersion=4.14.262-200.489.amzn2.x86_64,
remoteHostAddress=3.121.183.166, nodeId=n1}
```