BNWEngine Version Difference

Teh Min Suan

October 2024

1 Results

In this section, we present the test results of `BNWEngineV4.py`, focusing on its correctness and performance in comparison to `BNWEngineV2.py`. The output of `BNWEngineV4.py` is deemed correct if it matches the output of `BNWEngineV2.py`, with the exception of minor differences in timestamps or update times.

Key points for this section:

- RabbitMQ as a Non-Bottleneck:
  RabbitMQ is not a bottleneck in alert processing. All tests begin only after the RabbitMQ queue in BNWEngine is fully populated with alerts.
- Initial Cache State:
  The cache of `BNWEngine` starts off empty and is cleared before each test.
- Timer Start/Stop:
  The timer starts once the first output from `BNWEngine` is updated in the database and stops after X correct updates are made, where X represents the total number of alerts to be processed. Therefore, the recorded time reflects the processing duration for $X - 1$ alerts. Note that this excludes the startup time of `BNWEngine`, which benefits `BNWEngineV2`, as it has a significantly longer startup time compared to `BNWEngineV4`.
- Alert Processing Routes:
  - Route A:
    Alert information is sent to `listen_log`, prompting the `alerts_worker` to update the alerts collection in the database. This causes `BNWEngine` to retrieve and cache the alert data from the database.
  - Route B:
    Alert information is sent directly to RabbitMQ, bypassing the `alerts_worker`. This means the alert will not be updated in the database, and `BNWEngine` will not cache the information.

- Entity Name Handling:
  Alerts with repeated entity names are processed via Route A, while alerts with randomized entity names are processed via Route B.
- System Load:
  No other applications or scripts that could significantly impact machine performance are running during the tests.
- Test Environment:
  The tests are conducted on an Ubuntu machine with 8 logical cores and 16 GB of RAM.

1.1 Correctness of `BNWEngineV4`

To validate the behavior of `BNWEngineV4`, its output is compared with that of `BNWEngineV2`. The outputs are considered correct if the data being updated or inserted into the `test_timeline` and `riskscore_compiler` collections are identical.

We conducted the validation test using the following function:

```python
LINUX_EVENTS = [
    'ADD_USER', 'DEL_USER', 'MODIFY_USER', 'ADD_GROUP', 'DEL_GROUP',
    'MODIFY_GROUP', 'DEL_LOG', 'MODIFY_CONFIG', 'FAILED_SUDO', 'CHGRP',
    'CHOWN', 'CHMOD'
]

def correctness_test(loop):
    logger.info("Starting correctness test.")

    start_count0 = coll_timeline[0].count_documents({})
    start_count1 = coll_timeline[1].count_documents({})
    logger.info(f"start_count0: {start_count0}, start_count1: {start_count1}")

    for event_type in LINUX_EVENTS:
        command = [
            'python3', 'generate_logs_linux.py',
            '-C', 'linux_test', '-L', str(loop), event_type, '-AUID', 'dummyUser'
        ]
        subprocess.run(command, stdout=subprocess.DEVNULL,
        ↪  stderr=subprocess.DEVNULL)

    total_loop = loop*len(LINUX_EVENTS)
    end_count0 = coll_timeline[0].count_documents({})
    end_count1 = coll_timeline[1].count_documents({})

    while end_count1 - start_count1 < total_loop or end_count0 - start_count0 <
    ↪  total_loop:
```

```
            end_count0 = coll_timeline[0].count_documents({})
            end_count1 = coll_timeline[1].count_documents({})
        logger.info(f"end_count0: {end_count0}, end_count1: {end_count1}")

        riskscoreIsSame = check_riskscore()
        timelineIsSame = check_timeline()
        return riskscoreIsSame, timelineIsSame
```

The `correctness_test` function takes an argument, loop, and generates all the events in `LINUX_EVENTS` for the specified number of iterations. After updating the collections for the given loop iterations, it calls `check_riskscore` and `check_timeline` to verify whether the data in both collections are identical for `BNWEngineV4` and `BNWEngineV2`.

Some setup is required before running this function. Each `BNWEngine` updates different collections in the database, and these collections are cleared of data prior to each test. We ran the test multiple times with `loop = 10000`, and the results consistently showed that the output of `BNWEngineV4` matches that of `BNWEngineV2`.

1.2 Improvement of Multiprocessing

In this section, we highlight the improvement in processing speed achieved by using multiprocessing in `BNWEngine`.

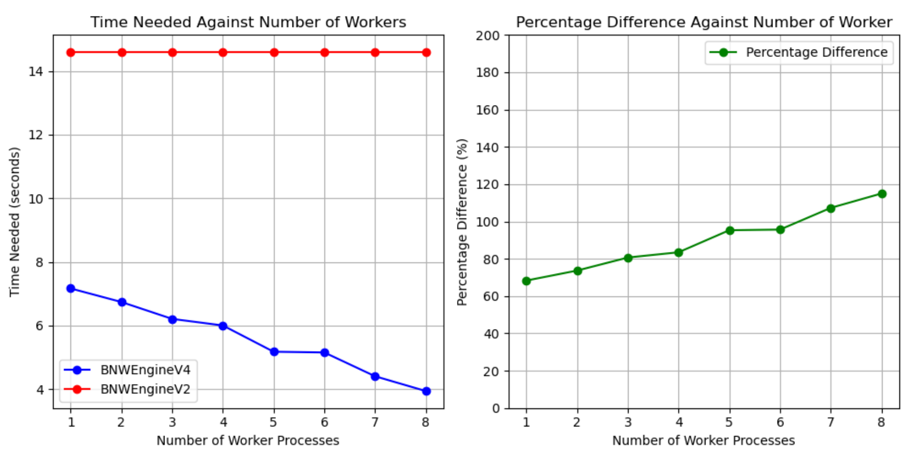

Figure 1: Sending Alerts via Route A and Increasing Worker Processes

Figure 1.2 presents the results of an experiment where 100 alerts were sent to both `BNWEngineV4` and `BNWEngineV2`. It is important to note that `BNWEngineV2` does not support multiprocessing, so increasing the number of worker processes does not enhance its performance.

In this test, both engines calculate the risk scores for entities without any historical alert data. To ensure this, the alerts are sent via Route A to prevent them from being cached. This setup minimizes any performance improvements that could result from better cache design, which will be discussed in Section 1.3.

As shown in the results, increasing the number of worker processes leads to a significant reduction in the time `BNWEngineV4` requires to process the 100 alerts, with improvements ranging from approximately 60% to 120%.

1.3 New Cache Design

In the tests described in this section, varying amounts of alert data are populated into the alerts collection in the database for each test. Additionally, the number of worker processes used by `BNWEngineV4` is limited to one, and alerts are sent via Route B.

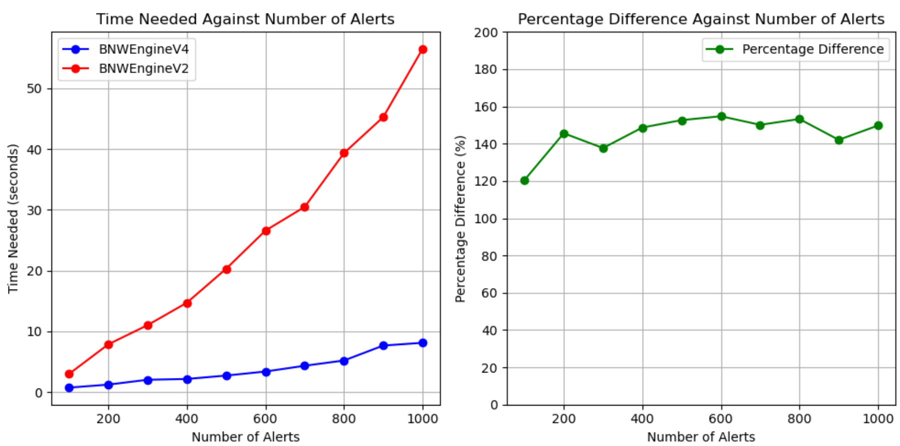

Figure 2: Sending Alerts via Route B

As shown in Figure 1.3, as the number of alerts increases, the difference in processing time between `BNWEngineV4` and `BNWEngineV2` becomes more pronounced. This is because more data needs to be retrieved and stored in the cache of `BNWEngine`. `BNWEngineV2`, with its inefficient cache design, requires significantly more processing time as the number of alerts increases.

In conclusion, the new cache design improves the performance of `BNWEngine` by approximately 120% to 160%.

1.4 Optimized Code

In this section, the number of worker processes used by `BNWEngineV4` is limited to one. Additionally, to minimize the performance improvement attributed to a better cache design, the alerts will be sent via Route A.

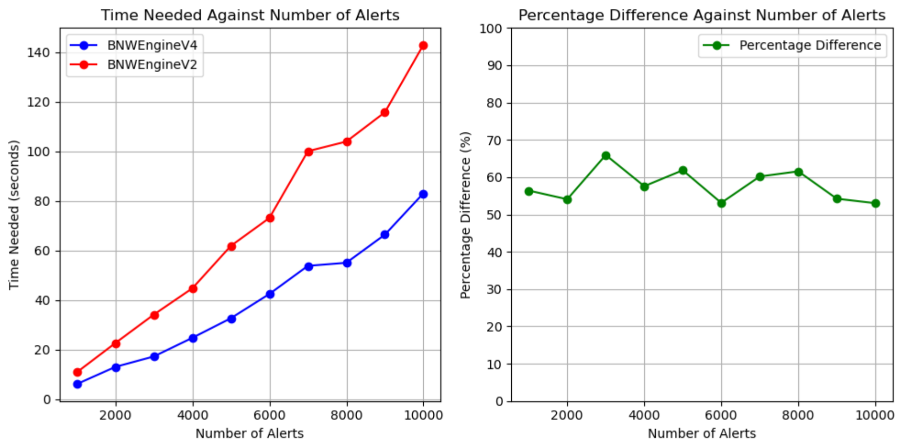

Figure 3: Sending Alerts via Route A and Increasing Alerts

As shown in Figure 1.4, as the number of alerts sent increases, the processing time required by both `BNWEngine` versions also increases. However, `BNWEngineV2` requires significantly more processing time than `BNWEngineV4`. Specifically, `BNWEngineV2` takes approximately 50% to 70% longer to process the alerts compared to `BNWEngineV4`.

1.5 Performance Difference Under Normal Circumstances

In this section, we will simulate real world scenario as close as possible. For instance, we will populate the database with 10000 alerts before running the tests. `BNWEngineV4` will create 8 worker processes as there are 8 logical cores in our machine. Then, we will send the alerts via Route A so that the alerts will be updated to database and cached in `BNWEngine`.

From Figure 1.5, we observe that the performance improvement of `BNWEngineV4` ranges from approximately 140% to 170%. We can confidently conclude that `BNWEngineV4` is indeed faster than `BNWEngineV2` under all circumstances, based on the findings in Section 1.

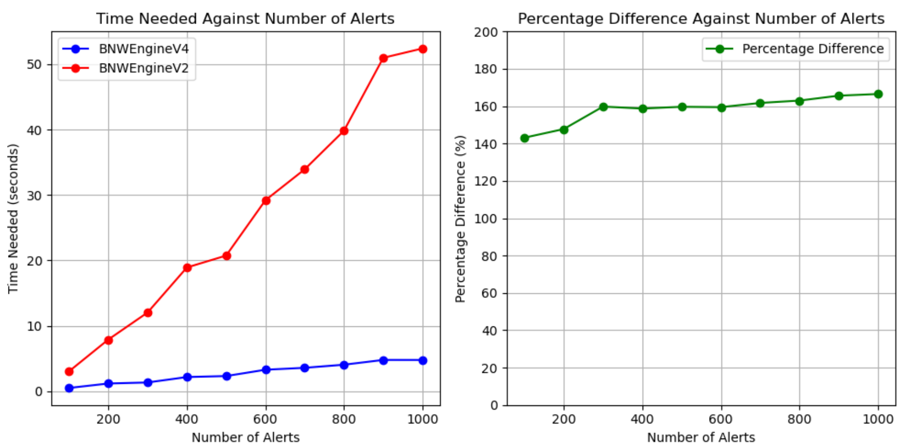

Figure 4: Simulation of Real World Scenario