Analytics Comparison

The following will compare performance between IOC (or DB2) and Mongo DB for a sample analytics task.

Liam (Cantwell) was implementing a POC where he wanted to calculate the number of alerts that are spatially and temporally related to a given alert. This was implemented using a standard DB2 stored procedure. We will look at how we might do the same thing using Mongo's aggregation pipeline (probably Mongo's strongest feature) and compare the performance of each.

IOC Version

This example SQL-PL procedure has been adapted to the weather events data source described here.

We pass in the ID of the event for which we want to find related events. We also pass in the number of minutes and distance within which events must fall to be considered related.

```
CREATE OR REPLACE FUNCTION IOC.EVENT_SMART_FEATURES (v_event_id INTEGER, v_number_minutes INTEGER,
v_distance_in_meters INTEGER)

    RETURNS TABLE (

            ID INTEGER,

            NUM_RELATED_EVENTS INTEGER)

    NO EXTERNAL ACTION

    LANGUAGE SQL

F1: BEGIN ATOMIC

    DECLARE v_number_minutes_each_side INTEGER;

    SET v_number_minutes_each_side = v_number_minutes / 2;

    RETURN

    SELECT E.INDEXNUM, COALESCE(HIT.NUM_EVENTS, 0) NUM_RELATED_EVENTS

    FROM IOC.TARGET_TABLE_WEATHER_EVENTS E

        LEFT JOIN

        TABLE (

            SELECT E1.INDEXNUM, COUNT(E2.INDEXNUM) as NUM_EVENTS

            FROM IOC.TARGET_TABLE_WEATHER_EVENTS E1, IOC.TARGET_TABLE_WEATHER_EVENTS E2

            WHERE

                E1.INDEXNUM = v_event_id AND

                E2.INDEXNUM != v_event_id AND

                E1.STARTDATETIME < E2.STARTDATETIME + v_number_minutes_each_side MINUTES AND

                E1.STARTDATETIME > E2.STARTDATETIME - v_number_minutes_each_side MINUTES AND

                db2gse.ST_Intersects(E2.LOCATION, db2gse.ST_Buffer(E1.LOCATION,
v_distance_in_meters, 'METER')) = 1

                            SELECTIVITY 0.00001

            GROUP BY E1.INDEXNUM

        )

        HIT

        ON E.INDEXNUM = HIT.INDEXNUM

    WHERE E.INDEXNUM = v_event_id;

END F1@
```

In order to iterate through the data source we create the following procedure. Since 1.6 million records would take to long to process we pass in the max number of events to process for the given test:

```
CREATE OR REPLACE PROCEDURE IOC.ALL_EVENT_SMART_FEATURES(V_MAX_EVENTS_TO_PROCESS INTEGER)
    LANGUAGE SQL
F1: BEGIN
    DECLARE STMT_TEXT VARCHAR(500);
    DECLARE EVENT_ID INTEGER;
    DECLARE BATCH_SIZE INTEGER DEFAULT 10000;
    DECLARE ITER INTEGER DEFAULT 1;
    DECLARE AT_END INT DEFAULT 0;
    DECLARE SQLCODE INTEGER DEFAULT 0;
    DECLARE retcode INTEGER DEFAULT 0;
    DECLARE NOT_FOUND CONDITION FOR SQLSTATE '02000';
    DECLARE STMT STATEMENT;
    DECLARE C1 CURSOR WITH HOLD FOR STMT;
    DECLARE CONTINUE HANDLER FOR NOT_FOUND SET AT_END = 1;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING SET retcode = SQLCODE;
    SET STMT_TEXT = 'SELECT INDEXNUM FROM IOC.TARGET_TABLE_WEATHER_EVENTS
                                        FETCH FIRST ' || V_MAX_EVENTS_TO_PROCESS || ' ROWS ONLY';
    PREPARE STMT FROM STMT_TEXT;
    OPEN C1;
    FETCH C1 INTO EVENT_ID;
    WHILE (ITER <= V_MAX_EVENTS_TO_PROCESS) DO
        INSERT INTO IOC.WEATHER_EVENTS_RELATED SELECT * FROM
TABLE(IOC.EVENT_SMART_FEATURES(EVENT_ID, 1000, 5000));
        IF (MOD(ITER,BATCH_SIZE) = 0) THEN
            COMMIT;
        END IF;
        FETCH C1 INTO EVENT_ID;
        SET ITER = ITER + 1;
    END WHILE;
    CLOSE C1;
    COMMIT;
END F1@
```

The full SQL-PL is attached here.

The output from the stored procedure goes into a table as follows:

IOC.WEATHER_EVENTS_RELATED

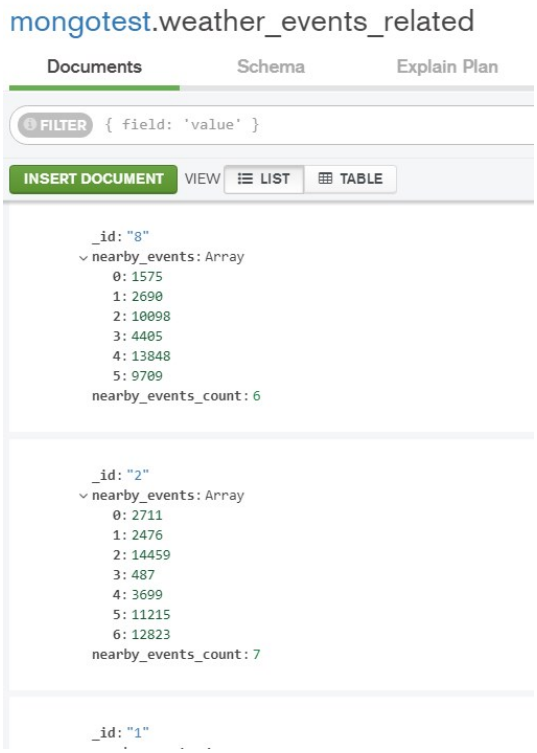
ID [INTEGER]	NUM_RELATED_EVENTS [INTEGER]
1	8
2	7
3	4
4	8
5	3
6	9
7	5
8	6
9	7
10	8
11	10
12	9
13	7
14	8
15	6
16	1

Mongo DB Version

The nearest equivalent to SQL-PL in Mongo is Javascript, which can be executed in the Mongo Shell. Since we can only execute the aggregation pipeline in this case a single event at a time (due to restrictions with the $geoNear stage) we will want to do the bulk event processing as close to the Mongo core as possible. So, we implement both procedures using js.

The method for calculating related events is shown below.

One enhancement over the SQL-PL version is that we also store an array of event IDs with each output record:




```
function get_nearby_events(event_id, event_loc, event_startdate, max_distance, max_minutes,
max_nearby_events = 100) {

    var minutes_either_side = max_minutes / 2;



    var output = db.weather_events.aggregate([

        {

            $geoNear: {

                near: event_loc,

                spherical: true,

                query: { startdatetime: { $lt : new Date(event_startdate.getTime() +
minutes_either_side * 1000 * 60),
```

```
                                            $gt : new Date(event_startdate.getTime() -
minutes_either_side * 1000 * 60) } },
                maxDistance: max_distance,
                num: max_nearby_events,
                distanceField: "dist.calculated"
            }
        },
        {
            $match: { indexnum: { $ne: event_id} }
        },
        {
            $project: {
                _id: 0,
                event_id: "$indexnum",
                centerpointid: event_id.toString()
            }
        },
        {
            $group: {
                _id: "$centerpointid",
                nearby_events: { $push: "$event_id" },
                nearby_events_count: { $sum: 1}
            }
        }
    ]);
    return output._batch.length > 0 ? output._batch[0] : no_nearby_events(event_id);
}
```

A restriction with the $geoNear pipeline stage is that it must come first. Thus we can only process a single event at a time in this case.

Two methods were tested for doing bulk event processing. The second batches the inserts into the output table:

```
function build_related_events_collection_method1() {
    var event_cursor = db.weather_events.find().limit(events_to_process);
    var cevent, nearby_events_record;

    db.weather_events_related.drop();
    while (event_cursor.hasNext()) {
        cevent = event_cursor.next();
        nearby_events_record = get_nearby_events(cevent.indexnum,
                                                 cevent.location,
                                                 cevent.startdatetime,
```

```
                                                max_distance,
                                                max_minutes,
                                                max_nearby_events);
        db.weather_events_related.insertOne(nearby_events_record);
    }
}
function build_related_events_collection_method2() {
    var event_cursor = db.weather_events.find().limit(events_to_process);
    var nearby_events_array = [];
    var cevent, nearby_events_record;
    var batch_size = 10000;
    var i = 0;

    db.weather_events_related.drop();
    while (event_cursor.hasNext()) {
        cevent = event_cursor.next();
        nearby_events_record = get_nearby_events(cevent.indexnum,
                                                cevent.location,
                                                cevent.startdatetime,
                                                max_distance,
                                                max_minutes,
                                                max_nearby_events);
        nearby_events_array.push(nearby_events_record);
        i++;
        if (i >= batch_size) {
            db.weather_events_related.insertMany(nearby_events_array);
            nearby_events_array = [];
            i = 0;
        }
    }
    if (i > 0)
        db.weather_events_related.insertMany(nearby_events_array);
}
```

The Javascript file for the above example is attached here. To execute we simply invoke mongo with the js file as parameter:

```
mongo ${pathToJsFile}/related_weather_events.js
```

To update a single event using the Mongo Java driver we can do something like the following:

```
public static void updateRelatedEvent_WeatherEvents(MongoTestDriver testDriver) {

        MongoCollection<Document> weatherEvents = testDriver.database.getCollection("weather_events");
        MongoCollection<Document> weatherEventsRelated = testDriver.database.getCollection("weather_events_related");
        Document aggregateDoc;
        Integer eventId = 6;
        Document event = null;
         int minutesEitherSide = 500;
         int maxDistance = 5000;
         int maxNearbyEvents = 100;
        Document noNearbyEvents = new Document("_id",eventId.toString())
                                                    .append("nearby_events",new ArrayList<Document>())
                                                    .append("nearby_events_count",0);
        UpdateOptions options = new UpdateOptions();
        options.upsert(true);

        event = findMatchingWeatherEvent(weatherEvents, "indexnum", eventId);

         if (event != null) {
               aggregateDoc = findNearbyWeatherEvents(weatherEvents, event, eventId, minutesEitherSide, maxDistance, maxNearbyEvents);

                 if (aggregateDoc != null) {
                    weatherEventsRelated.replaceOne(eq("_id",eventId.toString()), aggregateDoc, options);
               } else {
                    weatherEventsRelated.replaceOne(eq("_id",eventId.toString()), noNearbyEvents, options);
               }
       }
}

public static Document findMatchingWeatherEvent(MongoCollection<Document> weatherEvents, String idField, Integer id) {
      Document result = null;
      MongoCursor<Document> docs = weatherEvents.find(eq(idField, id)).limit(1).iterator();
       if (docs.hasNext())
            result = docs.next();
        return result;
}

public static Document findNearbyWeatherEvents(MongoCollection<Document> weatherEvents,
                                               Document event,
                                               Integer eventId,
                                               int minutesEitherSide,
                                               int maxDistance,
                                               int maxNearbyEvents) {
        AggregateIterable<Document> aggregateDocs;
        Document result = null;
        Document eventLoc = (Document)event.get("location");
```

```
        Date startDateTime = (Date)event.get("startdatetime");

        Date dateBefore = new Date(startDateTime.getTime() - minutesEitherSide * 1000 * 60);

        Date dateAfter = new Date(startDateTime.getTime() + minutesEitherSide * 1000 * 60);



        aggregateDocs = weatherEvents.aggregate(Arrays.asList(

                        new Document("$geoNear", new Document("near", eventLoc)

                                                        .append("spherical", true)

                                                        .append("query",

                                                                new Document("startdatetime",

                                                                        new Document("$lt",dateAfter)

                                                                                .append("$gt", dateBefore)))

                                                        .append("maxDistance", maxDistance)

                                                        .append("num", maxNearbyEvents)

                                                        .append("distanceField", "dist.calculated")),

                        match(ne("indexnum",eventId)),

                        project(fields(excludeId(),

                                        computed("event_id","$indexnum"),

                                       computed("centerpointid", eventId.toString()))),

                        group("$centerpointid",

                                push("nearby_events","$event_id"),

                                sum("nearby_events_count",1))

                        )

                );



        if (aggregateDocs.iterator().hasNext())

                result = aggregateDocs.iterator().next();



        return result;

}
```

Performance Comparison

	Events Processed	Time (s)	Rate (events per second)
IOC	250	400	0.625
Mongo - Windows			
Method 1	50,000	383	130
Method 2	50,000	347	144
Mongo - Linux			
Method 1	50,000	557	90
Method 2	50,000	481	103

As before (link), Mongo performance is significantly better than DB2. Bear in mind the output dataset is also richer in the Mongo case. Once again, Windows performs better than Linux for Mongo. mongostat (sample output shown below) shows the much higher insert rate in the case of Windows. The reason for the performance difference is yet to be determined. Perhaps Linux needs tuning to perform at its best.

Windows:

```
C:\mongodb\data>mongostat
```

```
insert query update delete getmore command dirty used flushes vsize  res qrw arw net_in net_out
conn                time

   124     *0     *0     *0       0   251|0  0.1% 9.2%       0 1.84G 654M 0|0 1|0   100k   92.2k    7
May 22 16:53:05.747

   127     *0     *0     *0       0   256|0  0.1% 9.2%       0 1.84G 654M 0|0 2|0   103k   93.2k    7
May 22 16:53:06.746

   126     *0     *0     *0       0   255|0  0.1% 9.2%       0 1.84G 654M 0|0 2|0   102k   92.8k    7
May 22 16:53:07.747

   127     *0     *0     *0       0   256|0  0.1% 9.2%       0 1.84G 654M 0|0 2|0   103k   93.8k    7
May 22 16:53:08.747
```

Linux:

```
[root@dubperf-mongodb weather_events]# mongostat

insert query update delete getmore command dirty used flushes vsize   res qrw arw net_in net_out
conn                time

    82     *0     *0     *0       0   166|0  0.0% 5.8%       0 3.16G 2.12G 0|0 1|0  65.8k   81.2k
2 May 22 16:54:08.926

    78     *0     *0     *0       0   159|0  0.0% 5.8%       0 3.16G 2.12G 0|0 2|0  63.0k   80.0k
2 May 22 16:54:09.929

    74     *0     *0     *0       0   150|0  0.0% 5.8%       0 3.16G 2.12G 0|0 2|0  59.4k   79.0k
2 May 22 16:54:10.928

    74     *0     *0     *0       0   150|0  0.0% 5.8%       0 3.16G 2.12G 0|0 2|0  59.3k   78.9k
2 May 22 16:54:11.926
```