# S15 15619 Project Phase 3 Report

**Performance Data and Configurations**

<table>
<tr><th>Best Configuration and</th><th>Results from the Live Test</th></tr>
<tr><td>Choice of backend (pick one)</td><td>MySQL</td></tr>
<tr><td>Number and type of instances</td><td>Live Test: 4 , m3.large instances</td></tr>
<tr><td>Cost per hour<br>(assume on-demand prices)</td><td>Live Test: 0.05</td></tr>
<tr><td>Queries Per Second (QPS)</td><td>INSERT HERE: (Q1,Q2,Q3,Q4,Q5, Q6)
<table>
<tr><td></td><td>Q1</td><td>Q2</td><td>Q3</td><td>Q4</td><td>Q5</td><td>Q6</td></tr>
<tr><td>score</td><td>139</td><td>0</td><td>14.24</td><td>119.2</td><td>16.74</td><td>36.71</td></tr>
<tr><td>tput</td><td>20822.7</td><td>17050.8</td><td>2543.8</td><td>7152.2</td><td>690.6</td><td>3670.6</td></tr>
<tr><td>ltcy</td><td>4</td><td>2</td><td>19</td><td>6</td><td>72</td><td>13</td></tr>
<tr><td>corr</td><td>100</td><td>0</td><td>56.00</td><td>100</td><td>97</td><td>100</td></tr>
<tr><td>err</td><td>0</td><td>100</td><td>43</td><td>0</td><td>3</td><td>0</td></tr>
</table></td></tr>
<tr><td>Rank on the scoreboard:</td><td>Phase 1: 39<br>Phase 2: 39<br>Phase 3: 33</td></tr>
</table>

**Team : MadHatters**
**Members : Sahibdeep Singh, Swati Chowdhury, Nandita Joshi**

**Rubric:**

**Each unanswered question = -10%**
**Each unsatisfactory answer = -5%**

**[Please provide an insightful, data-driven, colorful, chart/table-filled, and interesting final report. This is worth 20% of the grade for Phase 3. Use the report as a record of your progress, and then condense it before sharing it with us. Questions ending with "Why?" need evidence (not just logic)]**

**Task 1: Front end (you may/should copy answers from your earlier report-- each report should form a comprehensive account of your experiences building a cloud system. Please try to add more depth and cite references for your earlier answers)**

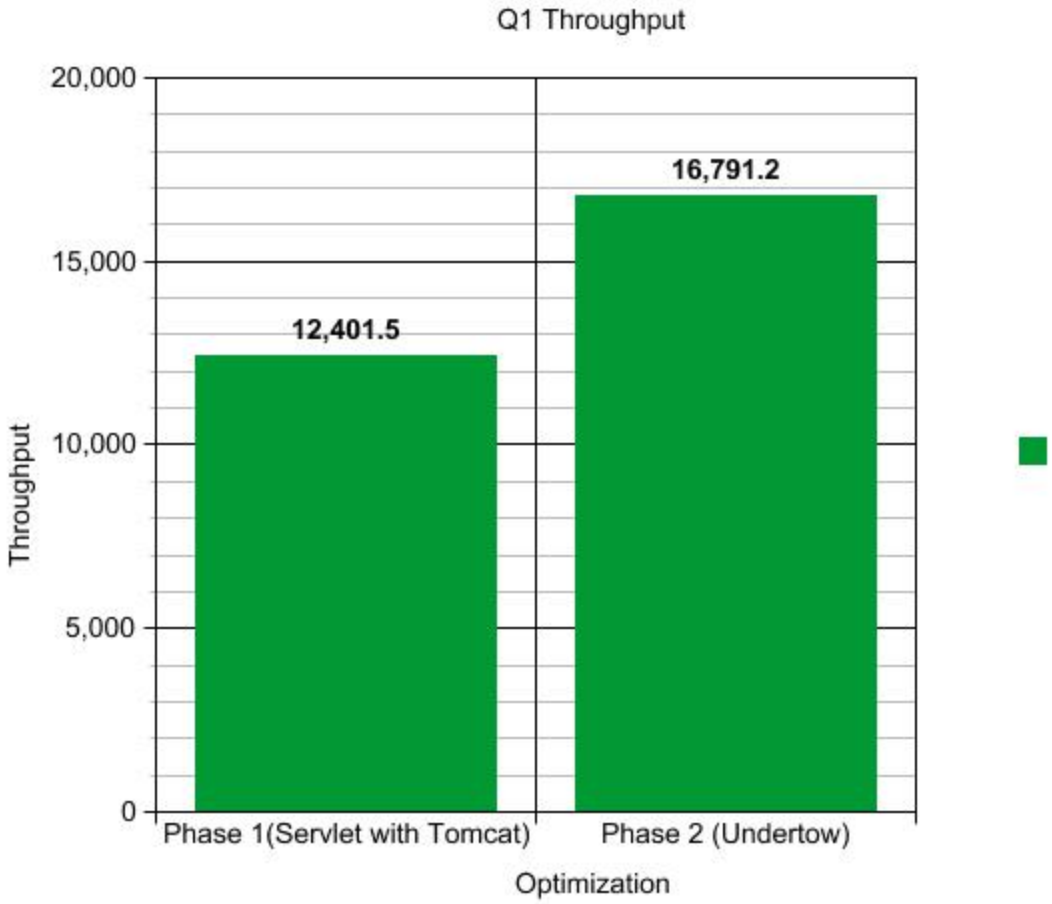
**Questions**

1. Which front end framework did you use? Explain why you used this solution. [Provide a small table of special properties that this framework/platform provides].

**Answer**: We used the Undertow framework as our frontend framework. For phase 1, we used servlets, but with Project 3.3 we had a
better understanding of Undertow and the dependencies to be used when deploying an application which used Undertow.
Experiment after Phase 1 ended:
Scenario: 1 ELB with three instances.
Output: Throughput of 16791.2 with a correctness of 100.0 and a latency of 4.
For Phase 2, Query 1:
Scenario: 1 m1.large instance.
Output: Throughput of 15317.3 with a correctness of 100.0 and a latency of 4.
Because of the stark difference in the architecture and the framework, we realised that working with undertow would make the throughput very high, while using less number of resources.
Table of properties for Undertow: [Extracted from undertow.io]
● Lightweight
Undertow is extremely lightweight, with the Undertow core jar coming in at under 1Mb. It is lightweight at runtime too, with a simple embedded server using less than 4Mb of heap space.
● HTTP Upgrade Support
Support for HTTP upgrade, to allow multiple protocols to be multiplexed over the HTTP port.
● Websocket Support
Undertow provides full support for Websockets, including JSR-356 support.
● Servlet 3.1
Undertow provides support for Servlet 3.1, including support for embedded servlet. It is also possible to mix both Servlets and native undertow non-blocking handlers in the same deployment.
● Embeddable
Undertow can be embedded in an application or run standalone with just a few lines of code.
● Flexible
An Undertow server is configured by chaining handlers together. It is possible to add as much or as little functionality as you need, so you don’t pay for what you are not using.

2. Did you change your framework after Phase 1 or Phase 2? Why or why not?

Answer: Yes we did. We used the Undertow framework as our frontend framework. For phase 1, we used servlets, but with Project 3.3 we had a better understanding of Undertow and the dependencies to be used when deploying an application which used Undertow. Experiment after Phase 1 ended:
Scenario: 1 ELB with three instances.
Output: Throughput of 16791.2 with a correctness of 100.0 and a latency of 4.
For Phase 2, Query 1:
Scenario: 1 m1.large instance.
Output: Throughput of 15317.3 with a correctness of 100.0 and a latency of 4.
Because of the stark difference in the architecture and the framework, we realised that working with undertow would make the throughput very high, while using less number of resources.



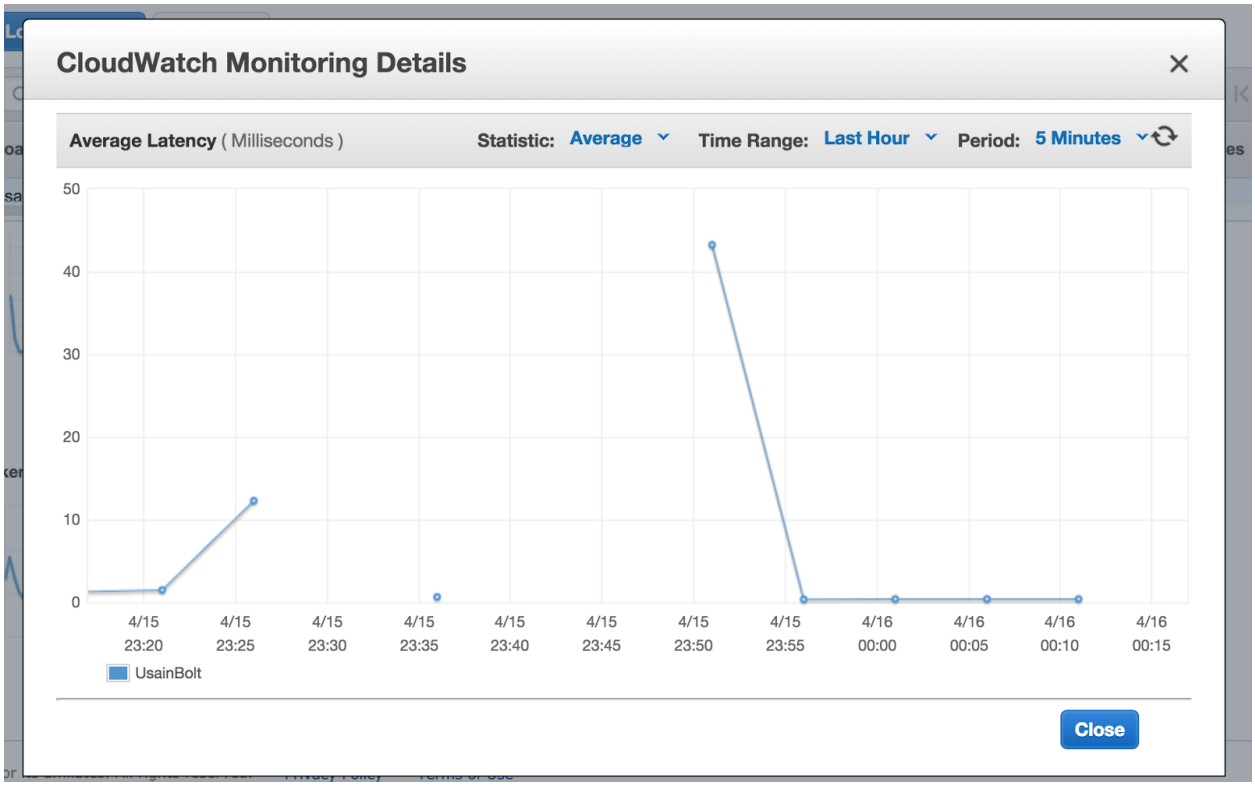
3. Explain your choice of instance type and numbers for your front end system.
   **Answer**:
   For our final submission we configured an Elastic Load Balancer, with three front-end instances of **Ubuntu Server 14.04 LTS (PV), SSD Volume Type** of m1.large and with root volumes of 20Gb each . As mentioned earlier, we chose to go ahead with m1.large instances as we could see that this combination of memory, cpu and storage could handle the flow of queries.

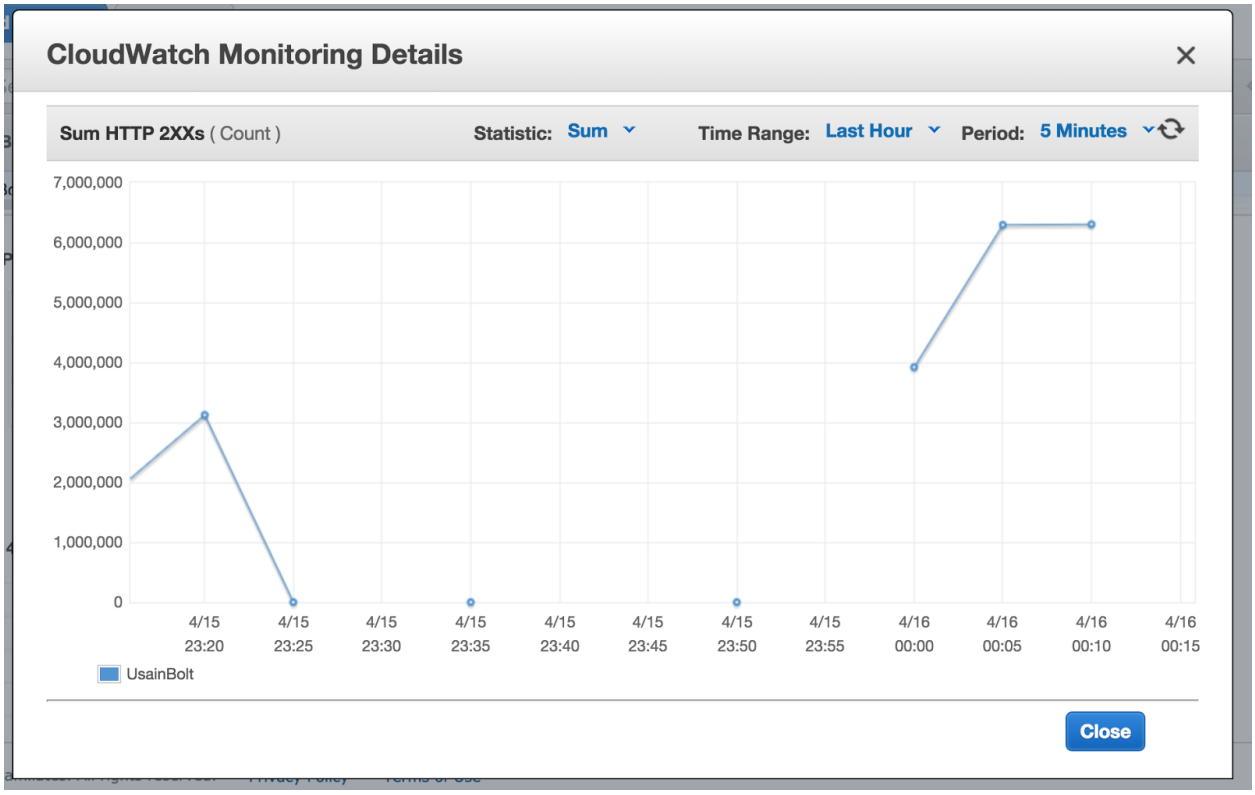
4. Explain any special configurations of your front end system.
   **Answer**: We used connection pooling in our front end to connect to MySQL.

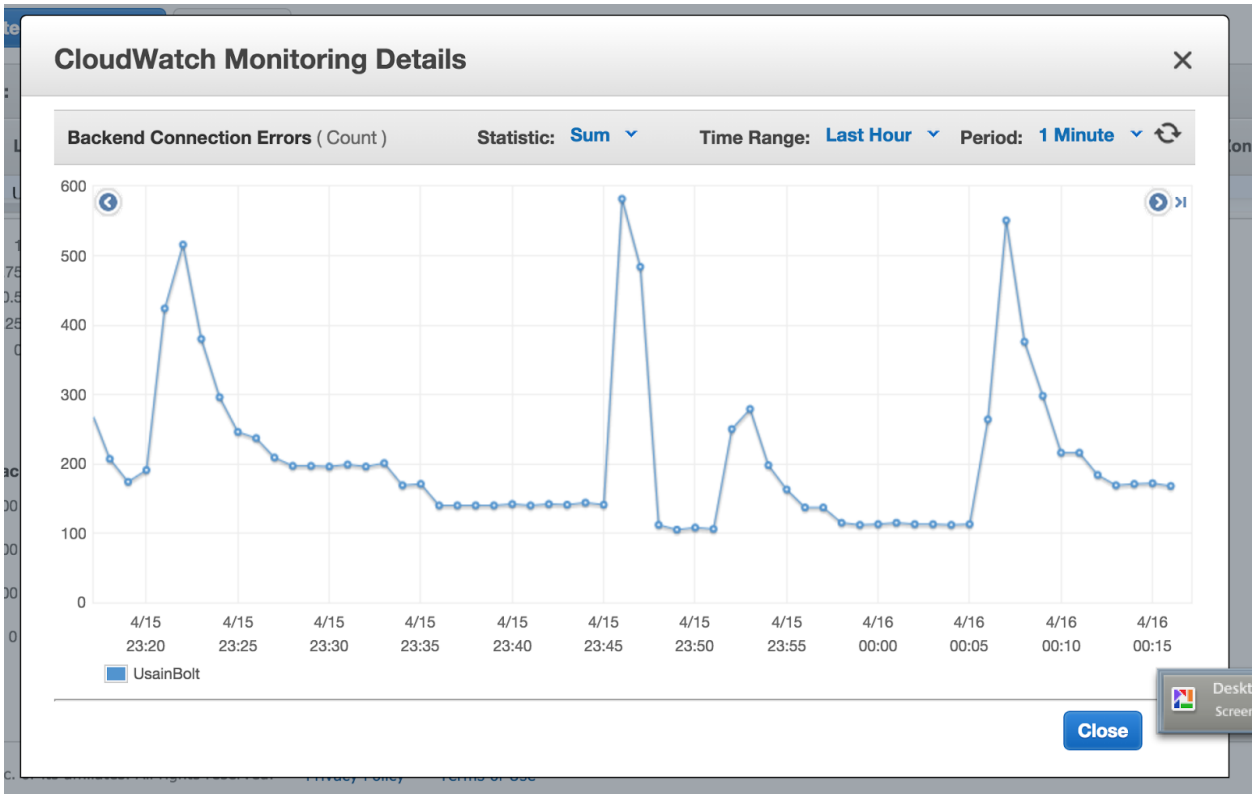
5. Did you use an ELB for the front-end? Why, or why not? Condense your experience with ELB in the next few sentences. Talk about load-balancing in general and why it matters in the cloud.
   **Answer**: Yes we did. Since we wanted load distribution across all instances and use the instances to their maximum potential, we used an ELB. Load balancing matters when the load is huge and we need multiple machines to manage those requests. ELB vastly improves the throughput by efficiently redistributing the requests on servers.
6. Did you explore any alternatives to ELB? List a few of these alternatives. What did you finally decide to use? (if possible) Provide some graphs comparing performance between different types of systems.
   **Answer**: No we only tried ELB.
7. Did you automate your front-end instance? If yes, how? If no, why not?
   **Answer**: No we did not automate our front end as we did not feel the need for it. We kept our front end very simple and basic in design.

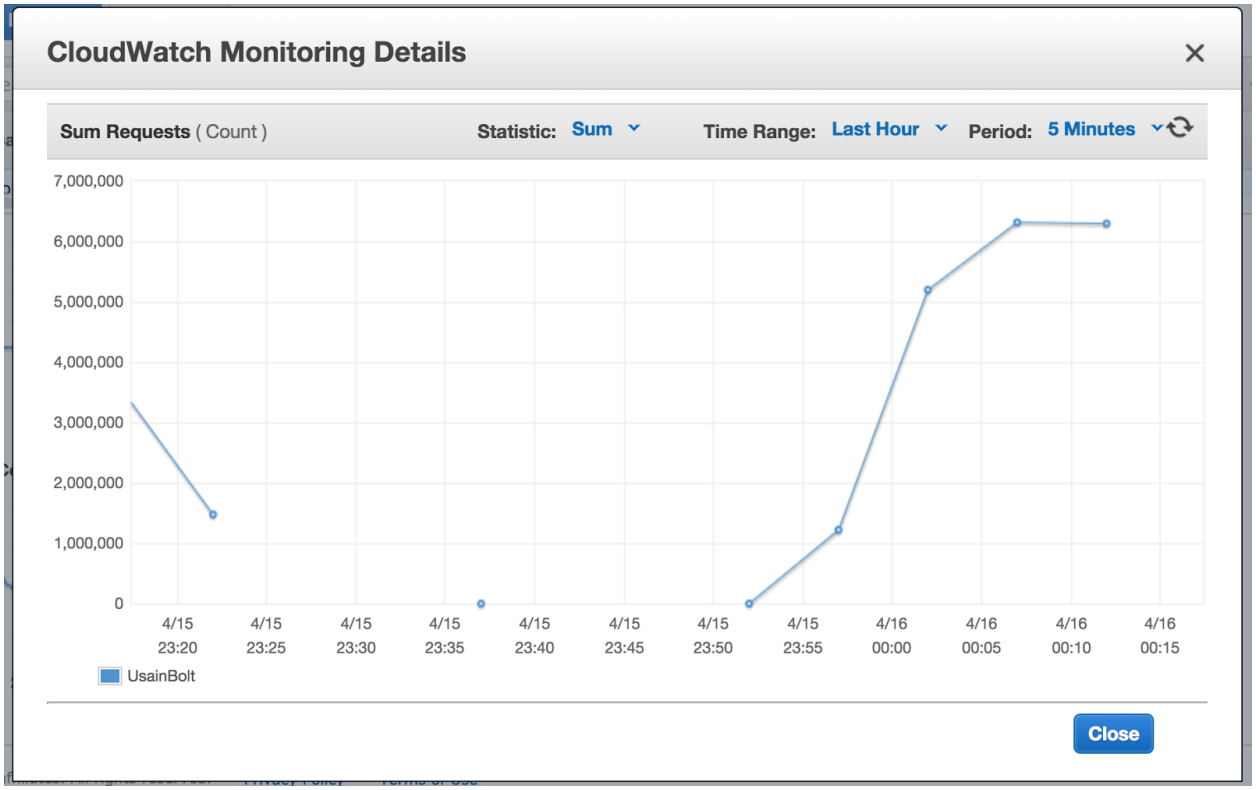
8. Did you use any form of monitoring on your front-end? Why or why not? If you did, show us a capture of your monitoring during the Live Test. Else, try to provide CloudWatch logs of your Live Test in terms of memory, CPU, disk and network utilization. Demarcate each query clearly in the submitted image capture.
   **Answer**: Yes we used cloud watch during the live test.

**Warm Up:[On our end, not during live test]**

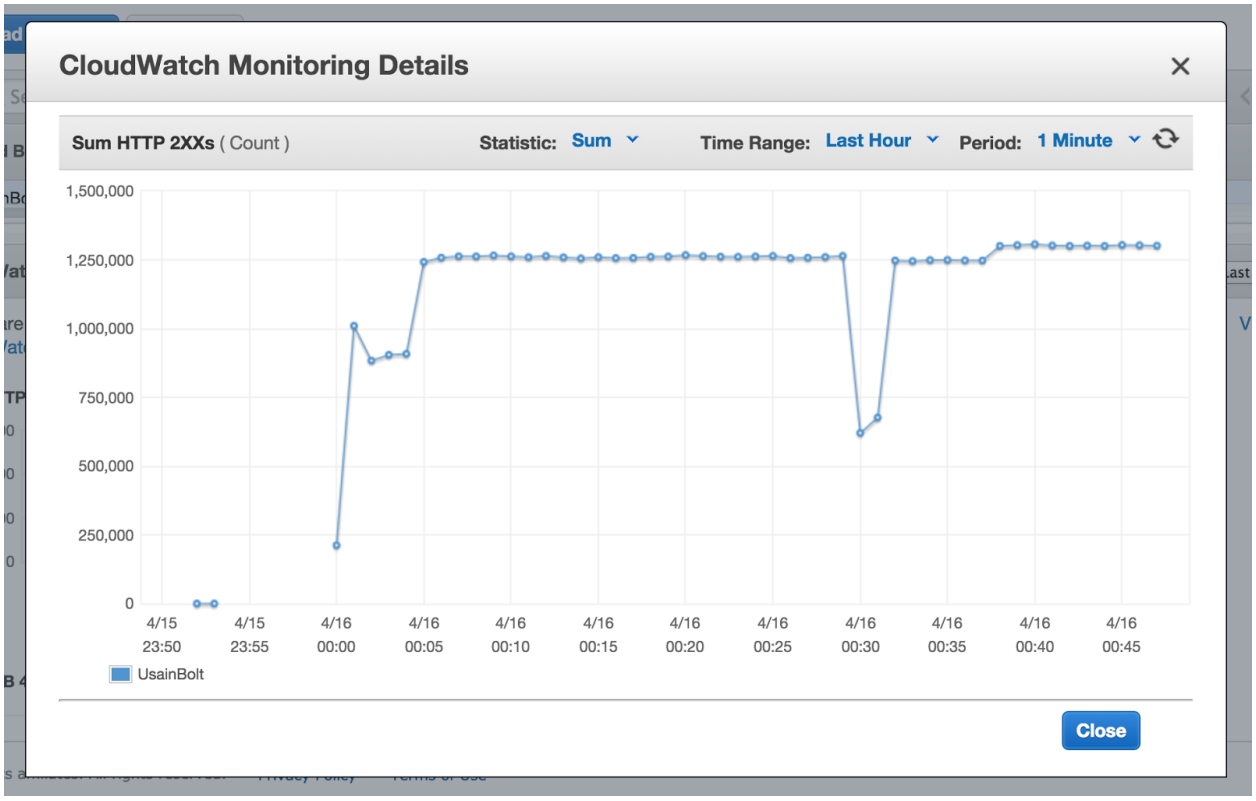


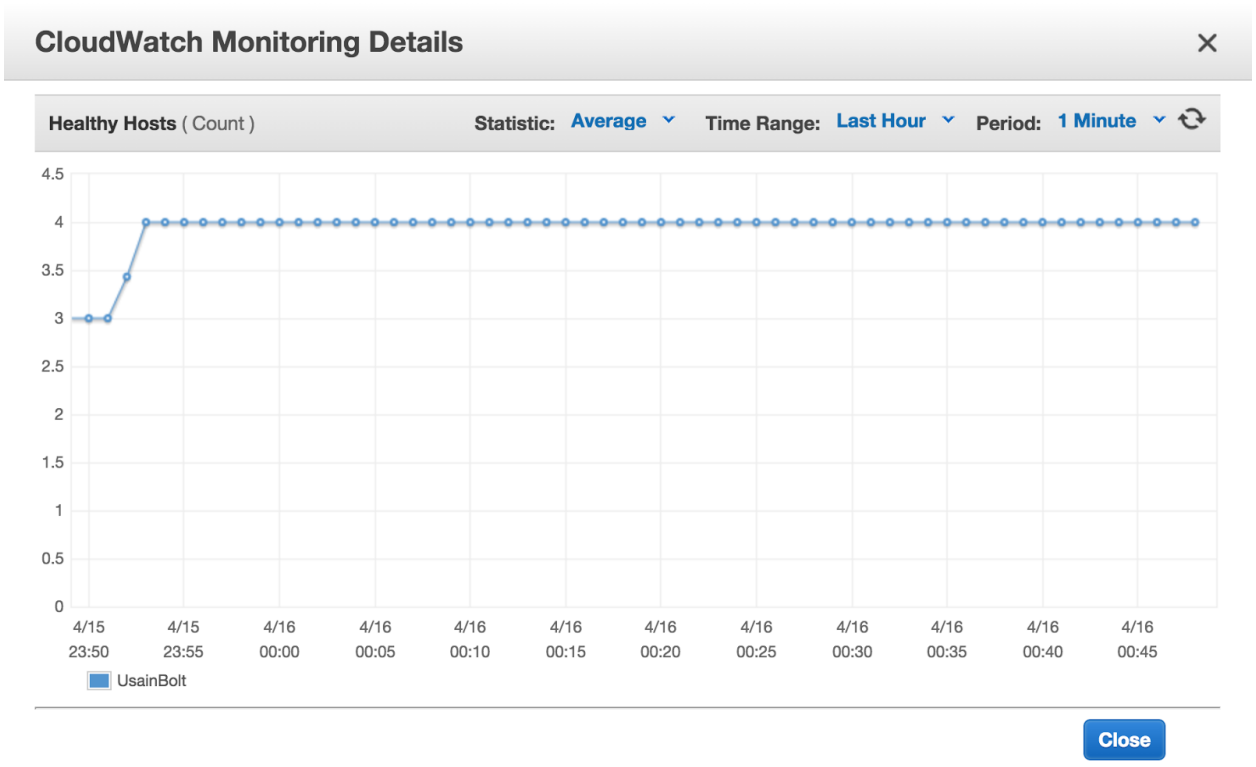

CloudWatch Monitoring Details
Backend Connection Errors ( Count )
Statistic: Sum
Time Range: Last Hour
Period: 1 Minute
600
500
400
300
200
100
0
4/15 23:20
4/15 23:25
4/15 23:30
4/15 23:35
4/15 23:40
4/15 23:45
4/15 23:50
4/15 23:55
4/16 00:00
4/16 00:05
4/16 00:10
4/16 00:15
UsainBolt
Close

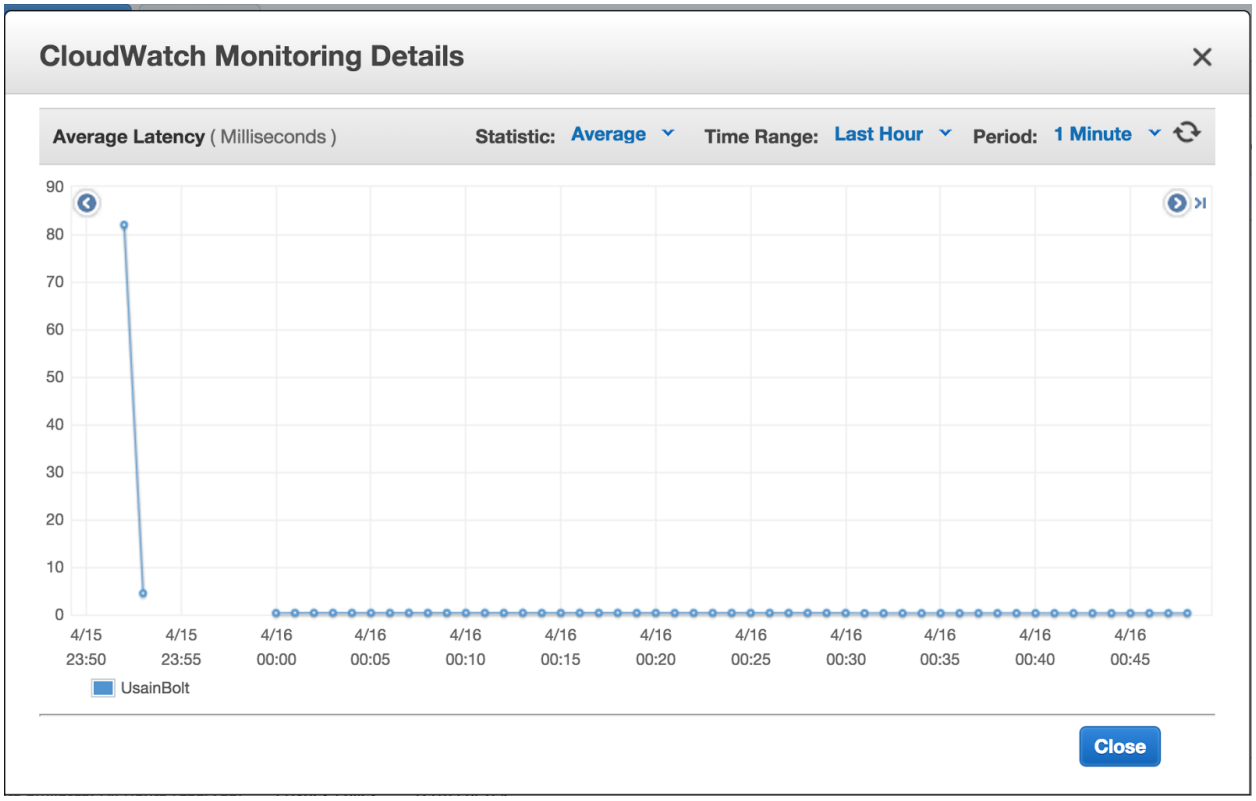

CloudWatch Monitoring Details
Sum Requests ( Count )
Statistic: Sum
Time Range: Last Hour
Period: 5 Minutes
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
4/15 23:20
4/15 23:25
4/15 23:30
4/15 23:35
4/15 23:40
4/15 23:45
4/15 23:50
4/15 23:55
4/16 00:00
4/16 00:05
4/16 00:10
4/16 00:15
UsainBolt
Close

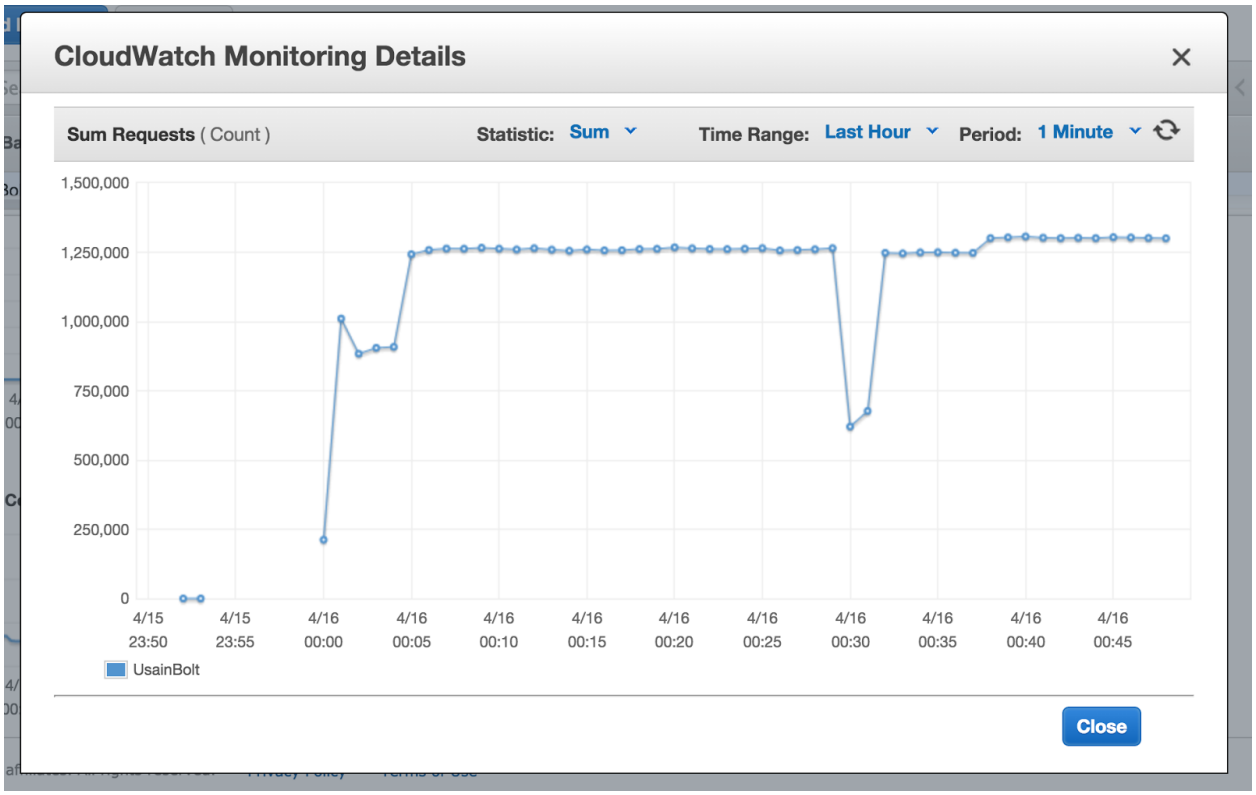
**During Live Test: [Warm Up Phase]**

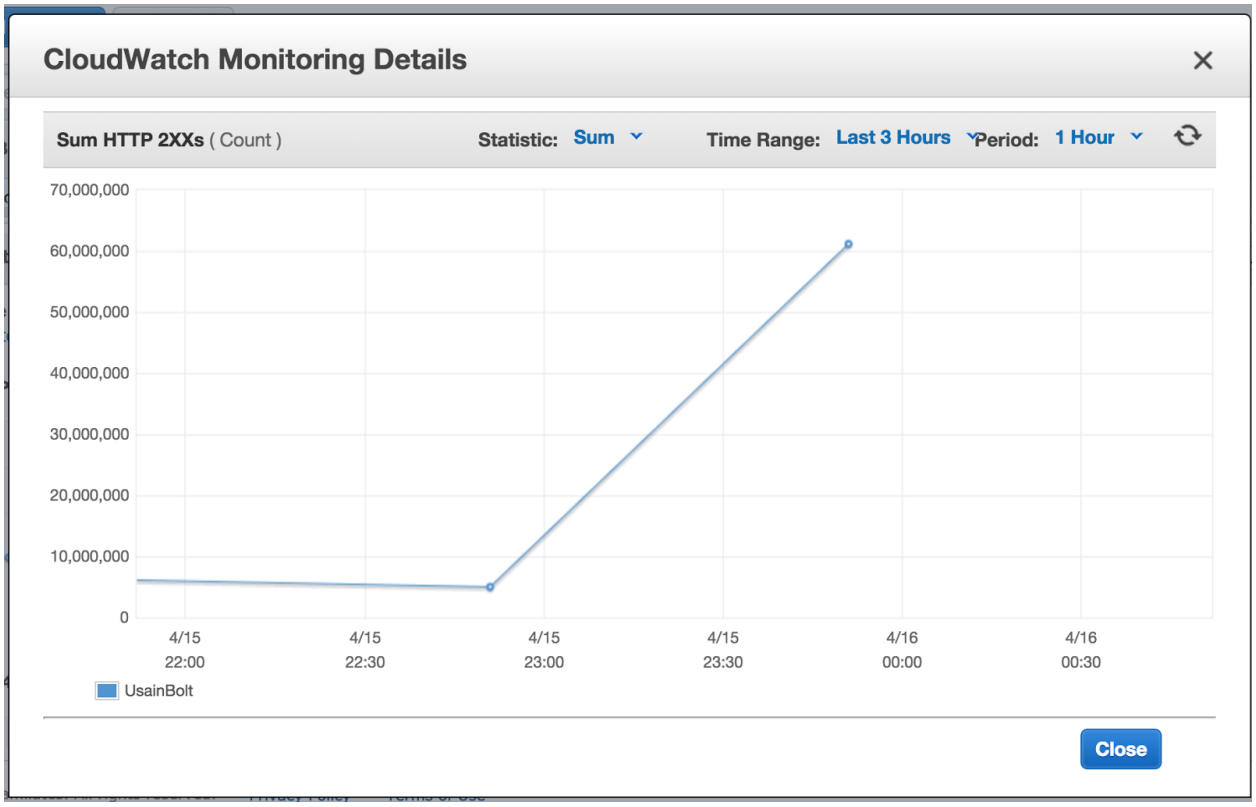



The drop is because of moving from the warm up to the actual query.

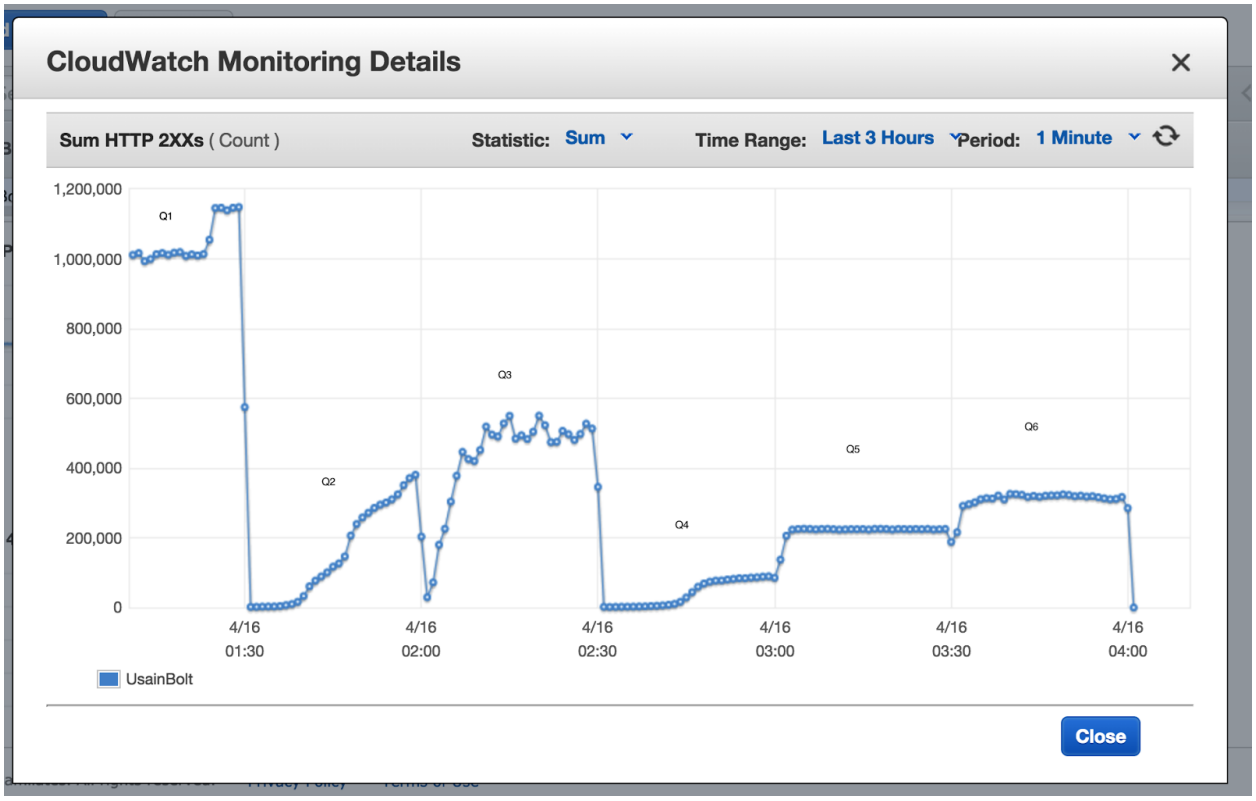


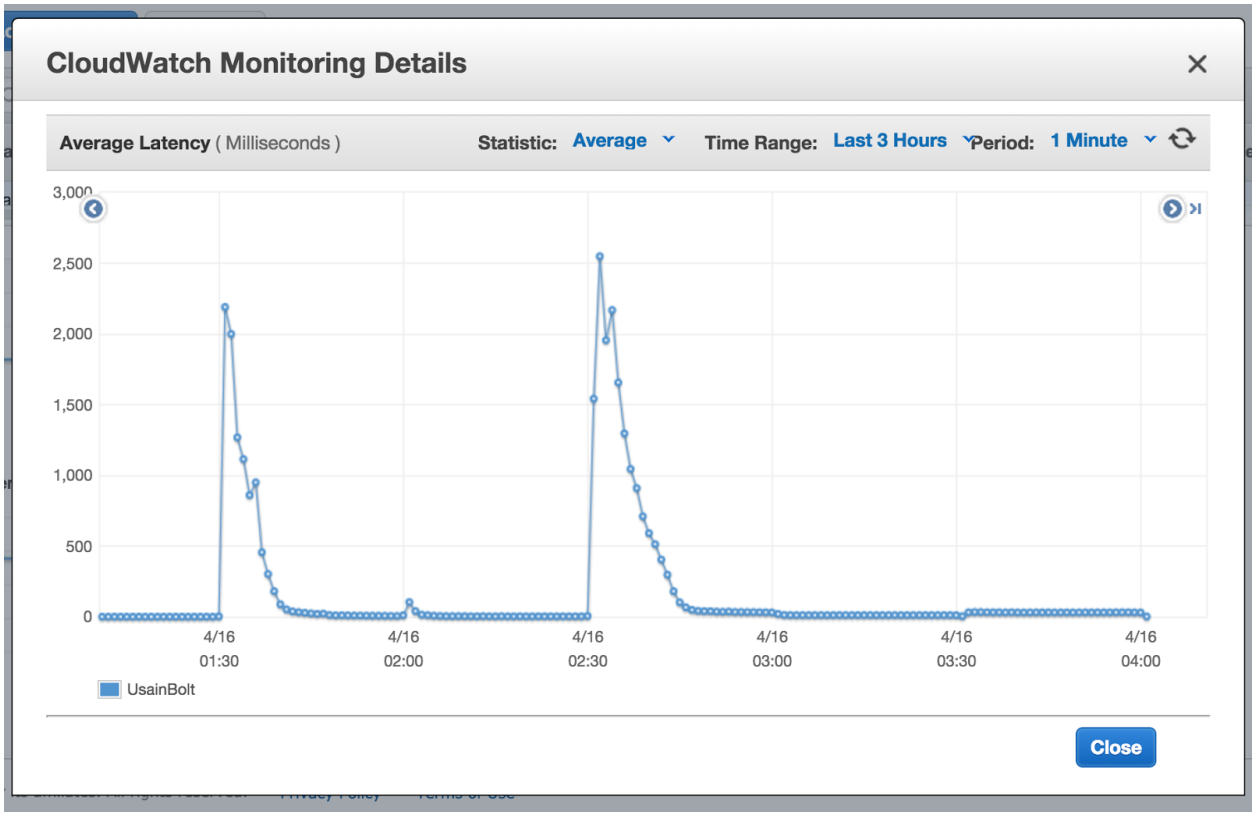

CloudWatch Monitoring Details
Average Latency ( Milliseconds )
Statistic: Average
Time Range: Last Hour
Period: 1 Minute
90
80
70
60
50
40
30
20
10
0
4/15 23:50
4/15 23:55
4/16 00:00
4/16 00:05
4/16 00:10
4/16 00:15
4/16 00:20
4/16 00:25
4/16 00:30
4/16 00:35
4/16 00:40
4/16 00:45
UsainBolt
Close


CloudWatch Monitoring Details
Sum Requests ( Count )
Statistic: Sum
Time Range: Last Hour
Period: 1 Minute
1,500,000
1,250,000
1,000,000
750,000
500,000
250,000
0
4/15 23:50
4/15 23:55
4/16 00:00
4/16 00:05
4/16 00:10
4/16 00:15
4/16 00:20
4/16 00:25
4/16 00:30
4/16 00:35
4/16 00:40
4/16 00:45
UsainBolt
Close

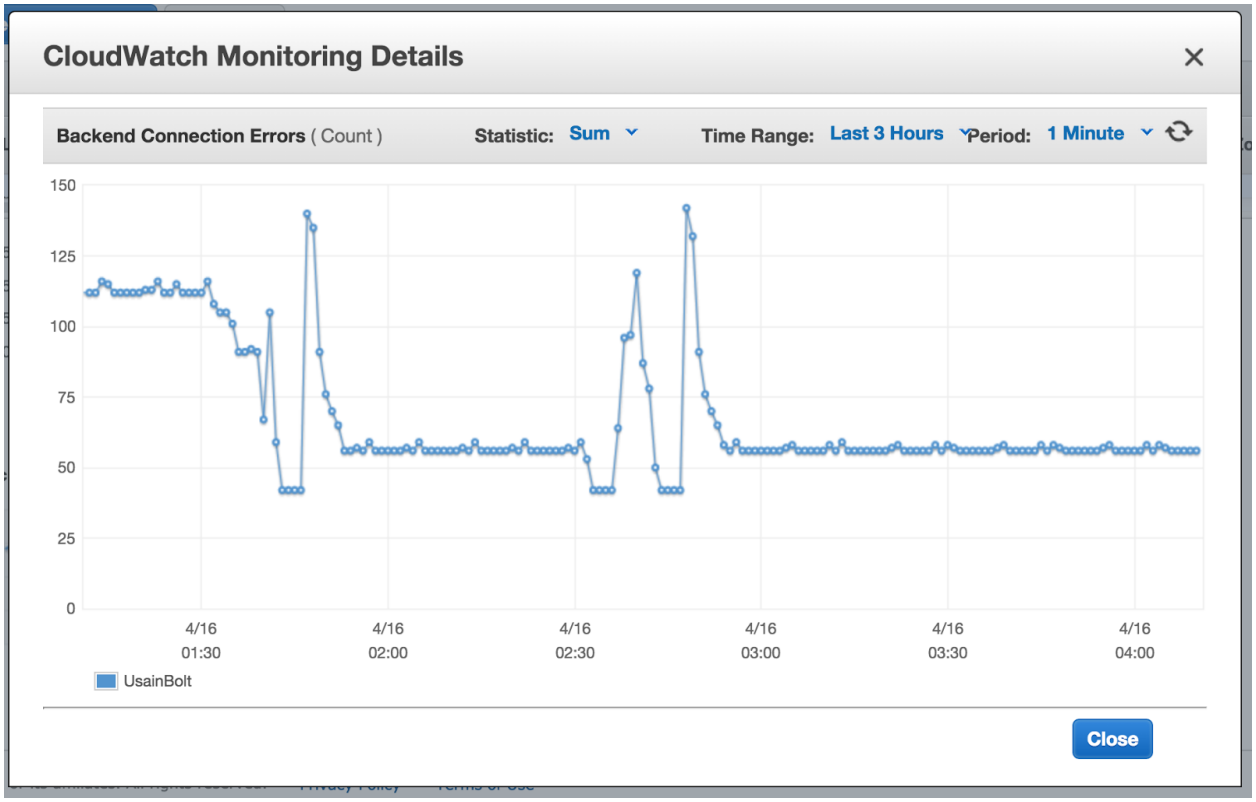

CloudWatch Monitoring Details
Sum HTTP 2XXs ( Count )
Statistic: Sum
Time Range: Last 3 Hours
Period: 1 Hour
70,000,000
60,000,000
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
0
4/15 22:00
4/15 22:30
4/15 23:00
4/15 23:30
4/16 00:00
4/16 00:30
UsainBolt
Close

## Post Test Summary:






Our backend connection errors spiked in the beginning but stabilized in a matter of milliseconds:




9. What was the cost to develop the front end system?
   **Answer**: Our cost of front end was 0.025 per hour. (4 m3 large * 0.05 +0.025 (elb cost))

10. What are the best reference URLs (or books) that you found for your front-end? Provide at least 3.
    - for undertow documentation, http://undertow.io/
    - for jdbc connection pooling: http://www.javatips.net/blog/2013/12/c3p0-connection-pooling-example?page=1
    - for connecting from front end to back end database: http://stackoverflow.com/questions/8348506/grant-remote-access-of-mysql-database-from-any-ip-address

Code for Front-End: Attached.

**Task 2: Back end (database)**

**Questions**

1. Which DB system did you choose in Phase 3? Why? Would any different queries for Q5 and Q6 have influenced you to choose the other DB?

**Answer**: We used MySQL as it was a collective team decision. We were more comfortable in tweaking the configurations for MySQL and at the same time well versed with connections, coding and configuration aspects on MySQL.

2. Describe your schema. Explain your schema design decisions. Would your design be different if you were not using this database? How many iterations did your schema design require? Also mention any other design ideas you had, and why you chose this one? Answers backed by evidence (actual test results and bar charts) **are required.**

   **Answer**:

   Design schema for various queries:

   q3: We designed the schema in 4 iterations. Initially we had planned to do all calculations in the front end. Therefore we only had user_ids. In the few next iterations we kept separate columns for +,-,* relationships. Finally we achieved the following schema -

| user_id | value |
|---|---|
| the user id queried by the front end | contained all the relationship between the user_id and other users |

q4: We designed the database for this query in 1 iteration only as it was very straightforward.

| hashtag | tweet_info |
|---|---|
| the hashtag queried by the front end | contained all the relevant tweet ids, user ids and timestamps for that hashtag |

q5: The database was designed in a single iteration. Schema was as below -

| user_id | timestamp | tweet_score | friends_count | followers_count |
|---|---|---|---|---|
| the user id queried by the front end | used for the start date and end date queried by the front end | contained the sum of all uniques tweets for a given timestamp | contained the maximum friends count of a user for a given timestamp, multiplied by 3 | contained the maximum followers count of a user for a given timestamp, multiplied by 5 |

In this schema, the tweet score, friends count and followers count values were pre-processed and stored for every day in order to reduce front-end query calculations.

q6**:** We designed the schema for this query in single iteration .

| user_id |
| --- |
| all the user_ids based on the logic for query 6. |

3. What was the most expensive operation / biggest problem with each DB that you had to resolve for each query? Why does this problem exist in this DB? How did you resolve it? Plot a chart showing the improvements with time.

**Answer**: for q4 we had this problem of case sensitivity. Since MySQL is not case sensitive by default we had issues setting up the primary key.
Apart from this, except for sizing the data types we did not have any particular problems. One of the most expensive operations for each query was indexing as that took a major chunk of time.

4. Explain (briefly) **the theory** behind (at least) 11 performance optimization techniques for databases. How are each of these implemented in MySQL? How are each of these implemented in HBase? Which optimizations only exist in one type of DB? How can you simulate that optimization in the other (or if you cannot, why not)? Use your own words (paraphrase, this document goes through plagiarism detection software). **(this question is worth 20 points if not answered in detail)**

**Answer**:

Database performance optimization techniques:
All of the techniques below are possible to implement in MySQL and HBase.

1. EXPLAIN - although this does not optimize performance by itself, using the EXPLAIN command on a query details the query execution plan - which is important in determining how the query is being optimized in the database.
2. JOIN - Using JOIN in place of SELECT loop queries reduces the total number of queries, making the search faster, thus improving performance.
3. WHERE - An index lookup is faster than a full table scan and this is achieved by using the WHERE keyword in a SQL query.
4. Caching - Front-end caching returns faster results. This is useful in our scenario where we have frequent reads and no real-time updates.
5. MySQL innodb settings - increasing the buffer pool size, log buffer size improves performance by providing greater storage and thus allowing for faster data processing.
6. Primary keys - Indexing by primary keys provides faster search. A primary key helps identify each row by a unique value.
7. Indexing - Indexing orders the data and improves performance of SELECT queries.
8. ORDER BY - Running ORDER BY on the data sorts it and limits the number of look-up

rows to those matching the conditions specified.
9. Sharding - Sharding improves write performance.
10. Replication - Replication makes reads faster.
11. Scaling - Scaling improves processing speeds.

5. Plot a graph showing results with/without each individual optimization that you used. Extremely impressive will be a timeline of rps v/s submission id (mentioning which optimization was in use at that time).

1. Indexing. We indexed the primary key columns in all our databases.
2. Increasing the max pool, max buffer sizes in my.cnf.
3. Data types and sizes. We kept the data types and sizes appropriate to the data.
4. Minimize the tables to save join operations. We only kept one table for both the queries we worked on.
5. Changing the bind address front 127.0.0.1 to 0.0.0.0 to allow external connections to the database.
6. Tried database sharding for q2 but the database loading was very slow. Instead used a single instance with 100gb volume attached and used “LOAD INFILE” for very fast
database insertion.
7. There were no Null values in any of the columns of our all tables.

We had the benchmarks like without indexing the query response time was close to 43 seconds in the case of q6. After indexing it came down to less than a second.

6. Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?

**Answer**: Yes our web service will support all CRUD operations. Since it is a generic Undertow web service anything will work in the relevant method body.

7. Which API/driver did you use to connect to the backend? Why? What were the other alternatives that you tried? What changed from Phase 1? From Phase 2?

**Answer**: We used JDBC to connect to backend. Except for the framework nothing else changed. We migrated to Undertow from servlets.

8. Can you quantify the speed differential (in terms of rps or Mbps) between reading from disk versus reading from memory? Did you attempt to maximize your usage of RAM to store your tables? How much (in % terms) of your memory could you use to respond to queries?

   **Answer:** Only thing we watch out was for memory usage using the command “df” and “df -h” to check the memory utilization. We made sure that the RAM was optimally utilized in the range of 40-50% and added 100GB to our ROM.

9. Did you use separate tables for Q2-Q6? How can you consolidate your tables to reduce memory usage?

**Answer**: Yes, we created different tables for all queries. Since we could only submit queries with databases in the phase 3 we did not get the time for table reusability.

10. What are the flaws you have seen in both DBs?

Answer: One of the major flaws in MySQL is the scalability compared to HBase. HBase was easier to scale and hence query retrieval has always been fast.
For HBase i think understanding how it works since it follows a different approach from traditional relational database and is more of an unstructured database, getting comfortable in that environment was a major issue.

11. How did you profile the backend? If not, why not? Given a typical request-response for each query (Q2-Q6) what percentage of the overall latency is due to:
    a. Load Generator to Load Balancer (if any, else merge with b.)
    b. Load Balancer to Web Service
    c. Parsing request
    d. Web Service to DB
    e. At DB (execution)
    f. DB to Web Service
    g. Parsing DB response
    h. Web Service to LB
    i. LB to LG

    **How did you measure this? A 9x5 (Q2 to Q6) table is one possible representation.**
    **Answer**: We did not profile our backend.

| | Q2 | Q3 | Q4 | Q5 | Q6 |
|---|---|---|---|---|---|
| **Load Generator to Load Balancer (if any)** | | 0 | 10 | 5 | 5 |
| **Load Balancer to Web Service** | | 10 | 10 | 10 | 5 |
| **Parsing request** | | 10 | 10 | 10 | 10 |
| **Web Service to DB** | | 10 | 10 | 10 | 10 |
| **At DB (execution)** | | 10 | 10 | 10 | 20 |
| **DB to Web Service** | | 20 | 20 | 20 | 10 |
| **Parsing DB response** | | 20 | 10 | 20 | 20 |
| **Web Service to LB** | | 10 | 10 | 10 | 10 |
| **LB to LG** | | 10 | 10 | 5 | 10 |
| | 100% | 100% | 100% | 100% | 100% |

12. What was the cost to develop your back end system?
    **0.2/hour**

What were the best resources (online or otherwise) that you found. Answer for HBase, MySQL and any other relevant resources. As we worked primarily on MySQL, the following links helped us the most:

- http://dba.stackexchange.com/questions/11806/why-is-drop-database-taking-so-long-mysql

- http://serverfault.com/questions/608870/error-iptables-no-chain-target-match-by-that-name
- http://stackoverflow.com/questions/29071072/java-net-connectexception-connection-refused
- http://stackoverflow.com/questions/13378566/cannot-connect-to-mysql-host-not-allowed
- http://stackoverflow.com/questions/8348506/grant-remote-access-of-mysql-database-from-any-ip-address

[Please submit the code for the backend in your ZIP file]

**Task 3: ETL**

1. For each query, write about:
    a. The programming model used for the ETL job and justification
    MapReduce
    b. The number and type of instances used and justification
    20 m1. large for quick data processing
    c. The spot cost for all instances used
    0.02/hour
    d. The execution time for the entire ETL process
    Approx 1 hour each query ->total 4 hours
    e. The overall cost of the ETL process
    $2.00
    f. The number of incomplete ETL runs before your final run
    Around 2-3 per query
    g. Discuss difficulties encountered
    During the ETL process, the major difficulty encountered was incorrect data processing which led to re run of the whole ETL job. Another issue faced was collating the data from ETL jobs and processing them into the database.
    h. The size of the resulting database and reasoning
    **q3: 3 gb (Approx), q4: 3.856 GB**
    **q6: 600Mb q5: 9.06Gb**
    i. The size of the backup
    We did not take the database backup instead we created snapshots and used them in AMIs. The snapshot size was 200GB.
2. What are the most effective ways to speed up ETL? How did you optimize writing to your backend? Did you make any changes to your tables after writing the data? How long does each load take?

**Answer**: The most effective way to speed up ETL we found was by making proper use of the MapReduce paradigm. Another important thing to consider was using data structures. We realized this towards 4 query. It is always better to reduce the data structures to minimum and avoid it altogether, if possible, for faster ETL jobs. Yes, we always added the index to our tables after data insertion. Each load took around 15-20 minutes after the query was executed to load data.

3. Did you use EMR? Streaming or non-streaming? Which approach would be faster and why?

**Answer**: We used a streaming EMR job as the data was read sequentially and it was easier to process using a streaming EMR job.

4. Did you use an external tool to load the data? Which one? Why?

**Answer**: No we did not use any external tools.

5. Which database has been easier to load (MySQL or HBase)? Why? Has your answer

changed in the last four weeks?

**Answer:** MySQL has always been our preferred choice. Although HBase was easier to load, our familiarity with MySQL operations always made it top choice for database system. No, our choice never changed.

[Please submit the code for the ETL job in your ZIP file]

**General Questions**

1. Would your design work as well if the quantity of data would double? What if it was 10 times larger? Why or why not?

**Answer:** Yes, our design is robust and it would be able to manage the load. Since we have kept ur design very simple and no database operations or data structures are used in the front end, it would be able to handle, process and manage the load.

2. Did you attempt to generate load on your own? If yes, how? And why?

**Answer:** Yes, we used one of the files on the server for testing with every query locally.

3. Describe an alternative design to your system that you wish you had time to try.

**Answer:** Not an alternative but we would've definitely tried caching specially for query 5 and query 6.

4. Which was/were the toughest roadblock(s) faced in Phase 3?

**Answer**:The toughest roadblock for Phase 3 was getting the throughput for query 5 and query 6. Since only in Phase 3 we had a complete team our first goal was to get the maximum queries running with 100% correctness and then worry about throughput. So, given the time our major road block was time because of which we could not try and experiment with the backend.

5. Did you do something unique (any cool optimization/trick/hack) that you would like to share with the class?

**Answer:** We implemented connection pooling as we observed that we continuously kept getting Socket errors. The rest of the tweaks were implemented in my.cnf for the MySQL database

**Bonus Questions**

1. Draw the Data Dependency DAG of fan-out requests (see P3.3 recitation slide 10 for an example of this graph) for the following sequence of events:
    a. Open the bonus page
    b. Search #beautiful  between 2014-01-01 and 2014-12-31
    c. Click on the first 10 tweets
    d. Click on the first 10 users

2. Did you use any parallelization to speed up this data fetching at the front-end? Did you use any front-end web technique to make it appear faster for a visitor to the web page?

3. Show the network graph from your browser when 1a-1d are performed.