The ELK Stack

Santiago J. Calcagno

Seminar “Big Data Tools” - Karlsruhe Institute of Technology

Table of Contents

Introduction

As time passes, it is increasingly necessary for the current businesses to get relevant information on the behavior of their clients in regards to the products and services they offer, as well as on the infrastructure supporting them, so they can obtain competitive advantages which keep them functioning. Such information, nevertheless, is usually submerged in a ever-growing sea of data, making it more difficult to extract by conventional means. Thus, advancements in tools and methods for extracting this information are crucial to obtain results that are useful in due time and form.

The following work will introduce the ELK or Elastic Stack, an open source tool which allows to acquire, analyze and visualize the aforementioned information coming mainly from system logs. Given a specific hardware platform, the effect of the variation of some of its running parameters on the performance of the stack will be analyzed. The Fundamentals section will describe each component of the stack in a succinct manner, emphasizing their role in it. The problem and the goals of this work will be defined in the Problem Description section. The Tests and Tools section will present the definition and structure of the tests, as well as the scripts and environments developed to execute them. These tests' results will be discussed in the Results section. To conclude, the possibility of further work and its viability in other hardware platforms will be discussed.

Fundamentals

Logs are an essential part of every software system. They allow us to get insights of the infrastructure's health, as well as how the users interact with such system, among others. These insights play an important role when it's time to take business decisions: they are crucial, for example, in the Lean Startup method, where certain metrics are what define progress in a startup [1].

As the amount of data flowing through an application rises, so does the volume, diversity, and velocity of its logs [2]. Thus, the process of analyzing them becomes more tedious, as each source needs to be investigated separately, and graphics need to be built with human interaction in order to summarize the general state of affairs. As this process does not take into account new logs coming at each instant, the insights that could have been obtained lose their relevance rapidly.

The ELK or, from now on, Elastic Stack is a set of software tools that "take data from any source, in any format and searches, analyzes, and visualizes it in real time" [3]. Managed by the open source vendor Elastic [4], it is made out of three separate projects: Elasticsearch [5], Logstash [6], and Kibana [7]. The stack is available as an on-premises application or as a service, deployable in the cloud [8].

Logstash

Logstash is an application which works as a pipeline for obtaining, processing, and outputting log data. Its potential relies in its amount of input, filter and output plugins that makes getting information coming from sources as diverse as system and app logs or social networks possible. In the stack, it is used to normalize the inputs and outputting them to Elasticsearch.

Elasticsearch

Elasticsearch is a search and analytics engine. It stores the output logs of Logstash and makes them available as JSON documents for Kibana to display them. It provides high scalability and resilience, as it can run in several nodes in distributed environments, and its performance on full text search makes it possible to obtain real time analytics.

Kibana

Kibana is a plugin for Elasticsearch that allows to create bar graphs, pie charts, maps, and several other statistical graphs. It is possible to create custom dashboards that display different information for different types of users, so that each one of them can get the insights that they need from the data.

Problem Description

In [9], the Logstash pipeline architecture is discussed. Figure 1 shows what the architecture was in versions 2.1 and older. The input, filter and output phases were separated by queues, serving as buffers.

Fig. 1. Logstash architecture, versions 2.1 and older.

As a bottleneck was noticed in the filter phase, with cpu-bound filters such as grok, date and geoip, the “w” flag was added to parallelize the filter workers, as shown in figure 2, given that most of the systems had more than one CPU core.

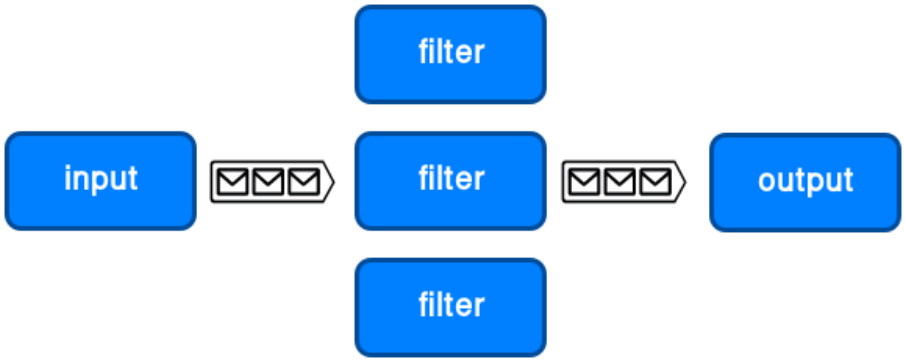

Fig. 2. Logstash architecture, versions 2.1 and older (more than one pipeline worker).

Later on, it was found that other bottlenecks existed. One example is the Elasticsearch index call in the output phase, which is bound to system I/O. Therefore, as figure 3 shows, output workers were introduced in Logstash 1.2.2. These outputs were managed by a worker proxy, which was injected when the amount of Elasticsearch output workers was set to a number greater than one. The proxy's function was to take events from its queue and dispatch them to each of the output workers as quickly as possible.

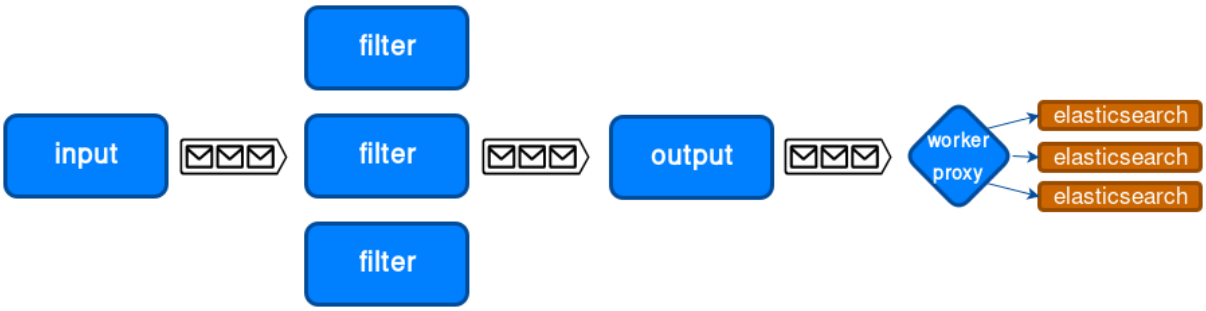

Fig. 3. Logstash architecture, versions 1.2.2 to 2.1.

In version 2.2 of Logstash, the queue between the filters and outputs was removed, seeing that the number of I/O operations in order to persist both queues would be excessive. Additionally, both filters and outputs were chosen to run in the same phase, with a previous batching stage. The "w" flag in this version defines the amount of pipeline workers running concurrently, each one with its own thread, and the "b" flag defines the batch size. As a design decision, the number of output workers was defined as default as the same as the amount of pipeline workers (see figure 4).

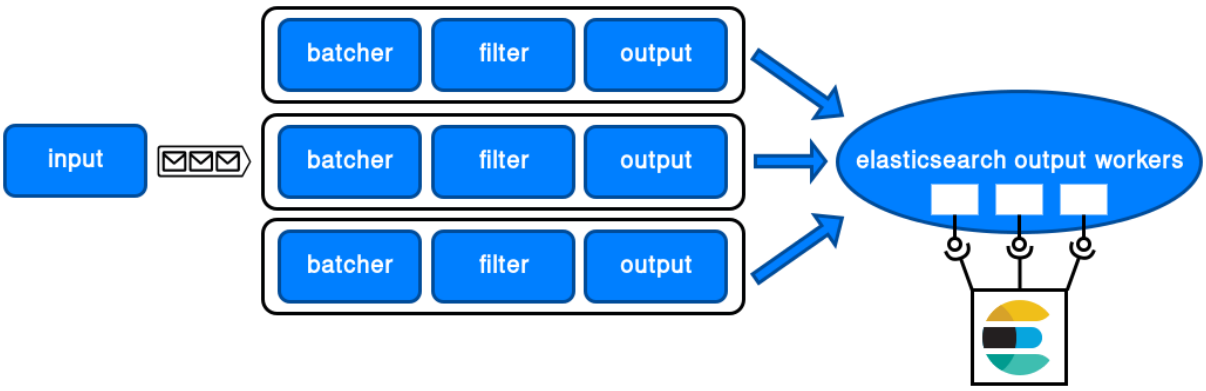

Fig. 4. Logstash architecture, version 2.2.

This, however, showed that a possible resource misuse would be eminent in the output workers, as each one of them managed concurrency with sufficient effectiveness by itself, and none of them shared resources with each other. The default number of output workers was then reverted to one in Logstash 2.3 and, in consequence, there was a reduction in throughput due to the fact that Logstash had less resources to use as a whole. The solution found for Logstash 5 was to make the output plugins "concurrency-friendly" or thread-safe (see figure 5).

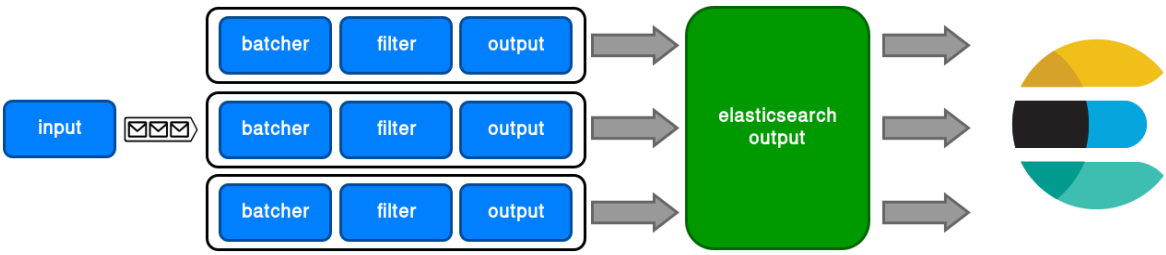

Fig. 5. Logstash architecture, versions 5.0 and newer.

Even though the selection of thread-safe output workers is now transparent for the user, the selection of the optimal number of pipeline workers and the batch size remains as a task to be done manually. As such, the goal of this work is to determine how do the variation of pipeline workers and the batch size affect the indexing rate in a given system.

Tests and Tools

With the goal of isolating the test environment's user space, a container image with the stack was used. At first, an image was created and modified based on [10], but after, the images available at [11] were used, as they were preferred due to their flexibility and support for Elastic's X-Pack. Such images are configured through the file "docker-compose.yml", where the number of pipeline workers and the batch size in Logstash's image is alternated as necessary for the tests. Additionally, the "LS_HEAP_SIZE" environment variable was set to 2048 MiB, so that the batch size modifications would have an impact on its performance.

As per the default values, Logstash's configuration file also allows logs to be passed through the TCP protocol, and sent to Elasticsearch. These values were left as they were, and the

filters were modified, adding one to parse JSON messages, and another to determine and store geolocation data in each entry. This was done to stress the CPU of the system as the tests ran.

The set of container images are controlled by a main script, “run.sh”, which starts and stops them, modifying the number of pipeline workers and the batch size in each iteration. Once the images are started and ready to receive data, the main script starts the entries generator (“jlog.py”) and the monitor (“gatherdata.sh”). When the entries generator finalizes, the monitor is closed, Elasticsearch’s database is cleaned, and the image is stopped. The process then repeats for the next pair of parameters.

The entries generator produces a JSON object whose fields are one of the previously defined server IP addresses, webservers, HTTP methods, queries, protocols, responses, and user classes. The script also generates a random username and a user IP, which will be processed by Logstash’s “geoip” filter to store geolocation data. The object is then converted to a string and sent via TCP to Logstash’s default port. This process is repeated until 500000 JSON objects are sent.

The monitor uses the program “curl” each second to make a request to Elasticsearch’s default IP and port, asking for the current index total. The standard output is then filtered to contain just the total index number, which is stored in the “data.txt” file.

The script “analyze.py” was used after the test was completed to calculate the average and the standard deviation of the result set found in the “data.txt” file.

To thoroughly test the effect of the variation of the amount of pipeline workers and the batch size in the system’s indexing rate, a wide range of values has been selected:

- Number of pipeline workers: from 1 to 16.
- Batch size (entries): 16, 32, 64, 125, 250, 500, 1000, 1250.

The hardware and software in which the tests were run is as follows:

- Processor: Intel® Core™ i5-2520M
- Memory: 16GB RAM DDR3-1866
- Hard disk: Samsung® EVO™ 250GiB mSATA SSD
- Operating system: Arch Linux, kernel 4.8.13-1
- Elastic stack version: 5.1

Results

Table 1 shows the results of the test runs, for batch sizes 16 to 125. It is important to remark that high values in the standard deviation are due to low initial performance, as well as Elasticsearch segment merging [12]. Here, a tendency can be seen, as the indexing rate improves as the batch size increases and there are more pipeline workers available. This is

especially the case when there are 125 entries per batch and the amount of workers is 2 or more.

Table 1. Test runs results, batch sizes 16-125.

	Batch size							
	16		32		64		125	
Workers	ø	σ	ø	σ	ø	σ	ø	σ
1	470,29	169,9	815,36	255,19	1234,49	368,81	1607,89	457,02
2	755,96	320,23	1390,37	499,35	2116,14	638,15	2648,12	750,88
3	814,72	366,14	1621,87	594,66	2460,57	763,73	2923,3	910,74
4	842,07	371,25	1611,88	594,62	2489,63	796,14	3000,48	1019,29
5	809,05	360,27	1600,24	586,46	2525,06	828,44	2958,56	931,13
6	838,98	374,51	1636,31	614,22	2541,44	797,43	2859,28	946,74
7	811,74	370,44	1584,6	604,3	2418,33	748,41	2852,3	955,84
8	808,33	370,77	1602,36	604,77	2533,09	786,65	2712,57	820,2
9	828,53	373,59	1657,72	584,81	2455,04	745,33	2668,47	818,61
10	819,48	186936,7	1652,18	582,14	2418,99	755,92	2671,45	787,96
11	811,27	362,56	1629,06	590,15	2368,29	723,37	2673	841,03
12	807,78	368,98	1616,89	607,44	2385,78	769,97	2594,67	770,05
13	822,2	368,54	1565,54	599,94	2412,29	740,41	2524,05	794,95
14	812,48	367,95	1576,21	590,63	2290,04	754,38	2527,95	824,68
15	778,72	358,81	1538,28	589,99	2361,38	788,14	2498,91	813,8
16	789,58	361,59	1621,75	580,69	2335,01	750,93	2494,66	838,12

In table 2, the average and the standard deviation results for the test runs with batch size 250 to 1250 are shown. The same observations made with table 1 can be made here, with the caveat that the batch size variation offers much less improvement in comparison with the increase of pipeline workers, until 3.

Table 2. Test runs results, batch sizes 250-1250.

	Batch size							
	250		500		1000		1250	
Workers	ø	σ	ø	σ	ø	σ	ø	σ
1	1966,54	523,41	2310,54	643,74	2446,07	690,83	2363,08	810,51
2	2965,53	866,45	2914,5	986,27	2760,75	1071,69	3006,47	1180,31
3	3039,87	1037,3	3128,39	1163,59	3032,66	1099,54	3070,07	1191,51
4	3030,54	1068,48	3002,73	1138,05	3064,3	1254,04	2984,44	1304,37
5	3030,3	936,33	3054,72	1093,34	3077,1	1222,73	3061,18	1479,35
6	2997,1	953,67	3009,34	932,32	3098,8	1157,89	3068,35	1471,81
7	2899,93	971,05	3003,11	1093,26	3064,52	1282,6	3046,8	1268,03
8	2872,77	933,8	3061,86	981,92	3031,78	1218,13	3135,24	1393,1
9	2880,57	915,44	3004,9	1112,84	3048,89	1222,2	3080,6	1321,04
10	2917,63	912,65	2974,23	1021,19	3038,13	1369,05	3064,47	1525,83
11	2834,55	909,69	2955,3	1048,35	2973,62	1240,42	3041,58	1397,4
12	2823,17	916,45	2970,33	1133,52	2964,74	1385,51	3054,46	1448,26
13	2736,3	914,8	2902,65	1070,01	3047,03	1239,66	3014,78	1283,67
14	2762,22	893,61	2919	1107,13	2914,45	1288,3	2959,92	1384,5
15	2741,56	911,79	2878,61	1111,33	2907,58	1429,88	2921,42	1268,64
16	2755,68	909,13	2896,58	1035,28	2941,89	1227,43	2964,36	1244,43

Figure 6 shows a comparison between the different batch sizes, and how they affect the average indexing rate, depending on the number of pipeline workers available. Here, it is clearly seen that there are significant improvements on the average indexing rate when this parameter is set to 125 (Logstash's default value [13]) or greater. The differences are less noticeable with higher values, arguably due to the limit of the system's CPU and disk.

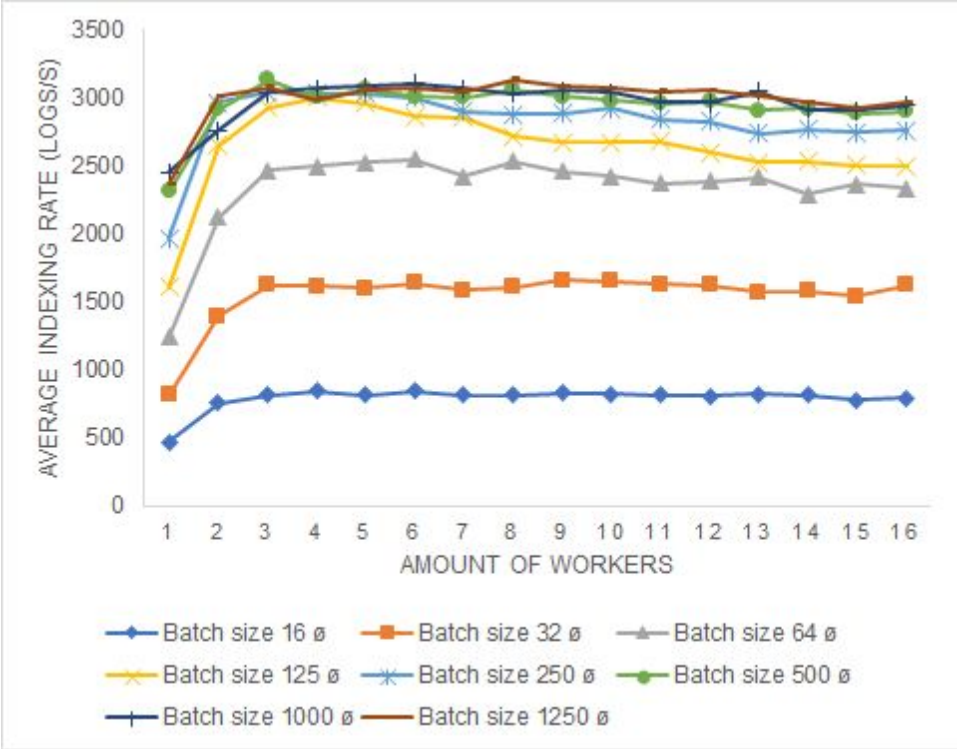

Fig. 6. Batch size comparison.

Figure 7 shows the same data, comparing the pipeline workers. It is possible to infer that, with the exception of the one-pipeline-worker setting, the availability of more memory compensates for the lack of more concurrency in the case of a low amount of workers. There are, however, cases where the peak near 3000 average indexed logs per second is reached with no big amounts of RAM, such as 4 workers with a batch size of 125, or 3 workers with a batch size of 500.

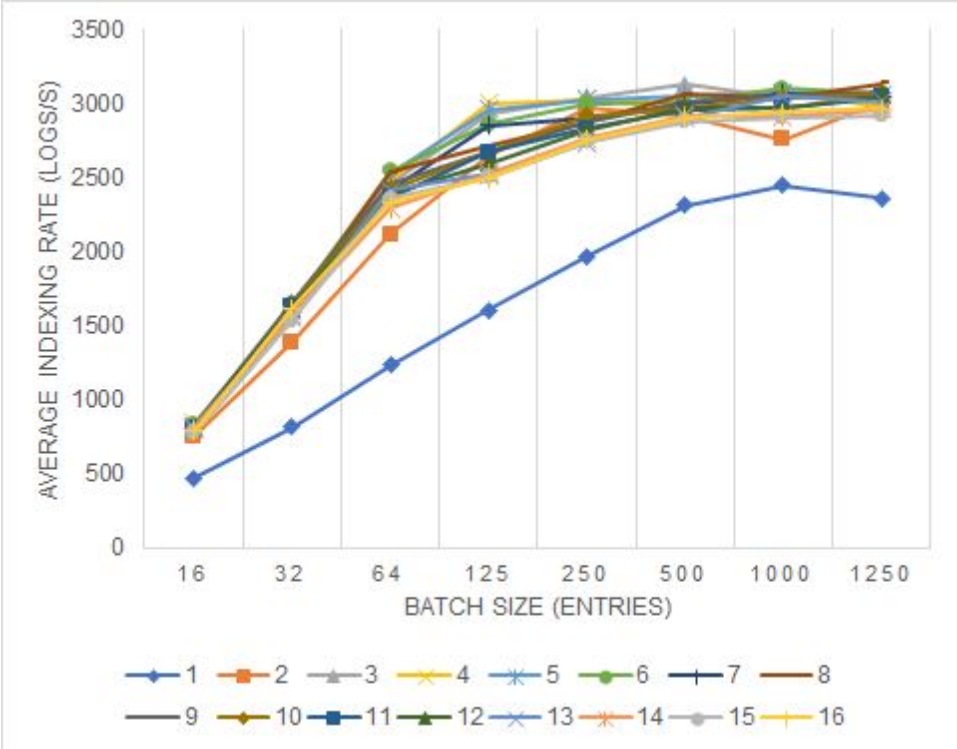

Fig. 7. Output workers number comparison.

Special cases can also be described. For example, figure 8 shows how, no matter which number of workers are available, the average indexing rate with a batch size of 16 entries is capped near 850 indexes per second.

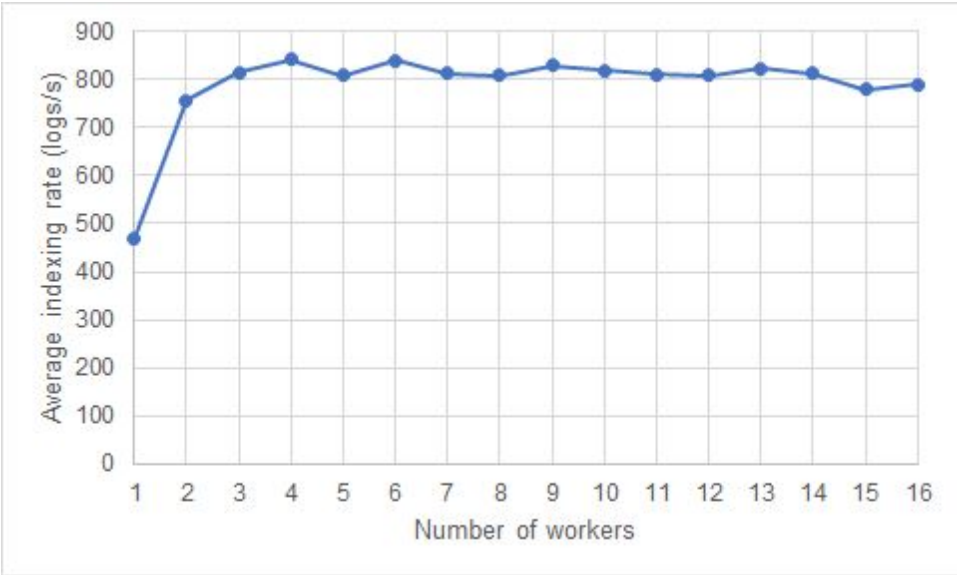

Fig. 8. Indexing rate (16 entries per batch).

As per figure 9, this limit approximately doubles when the batch size is doubled.

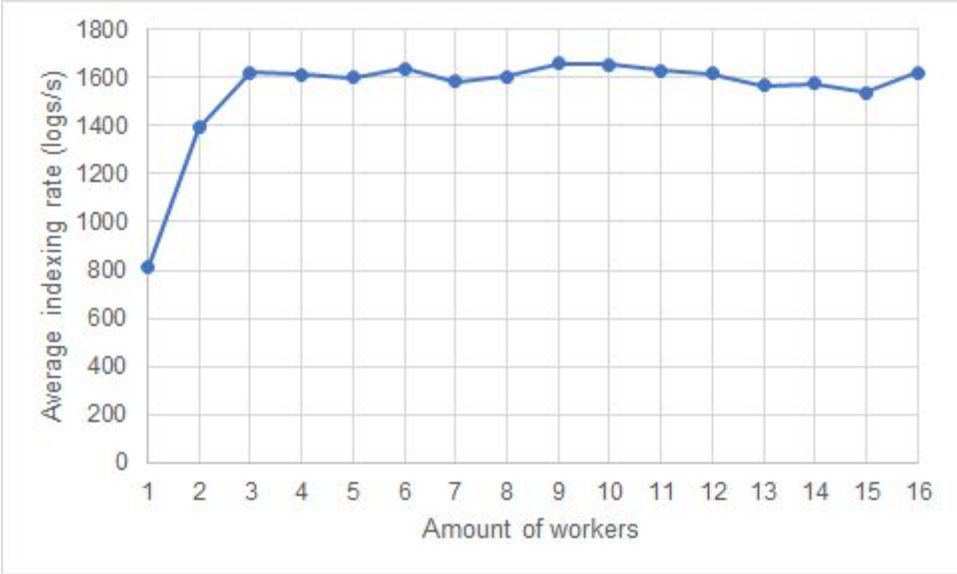

Fig. 9. Indexing rate (32 entries per batch).

With the default 125 entries per batch, the cap nearly doubles once again to system limits, with the average indexing rate dropping as the available workers go past 4 (see figure 10).

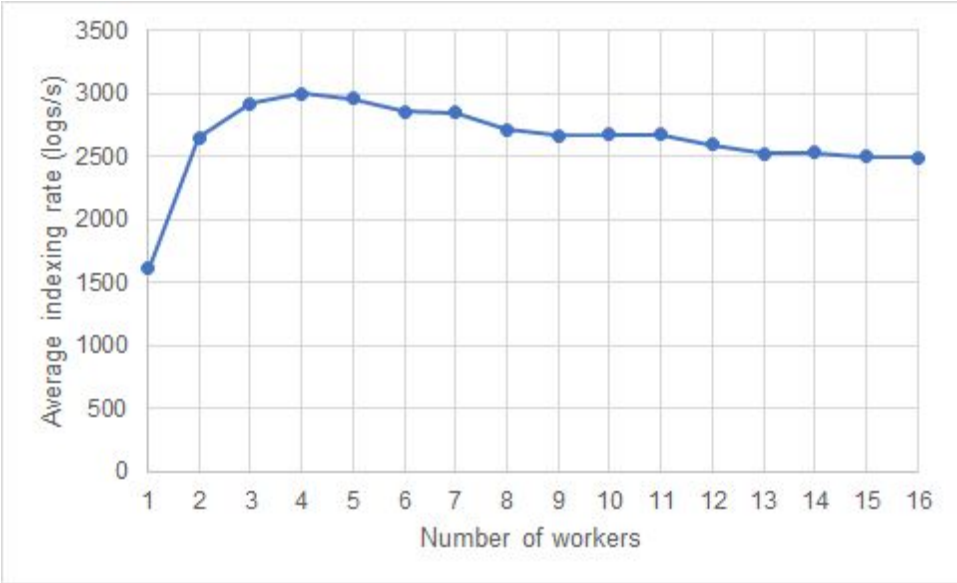

Fig. 10. Indexing rate (125 entries per batch).

This drop can be compensated increasing the available memory, with graphs similar to figure 11 with a batch size of 250 or greater.

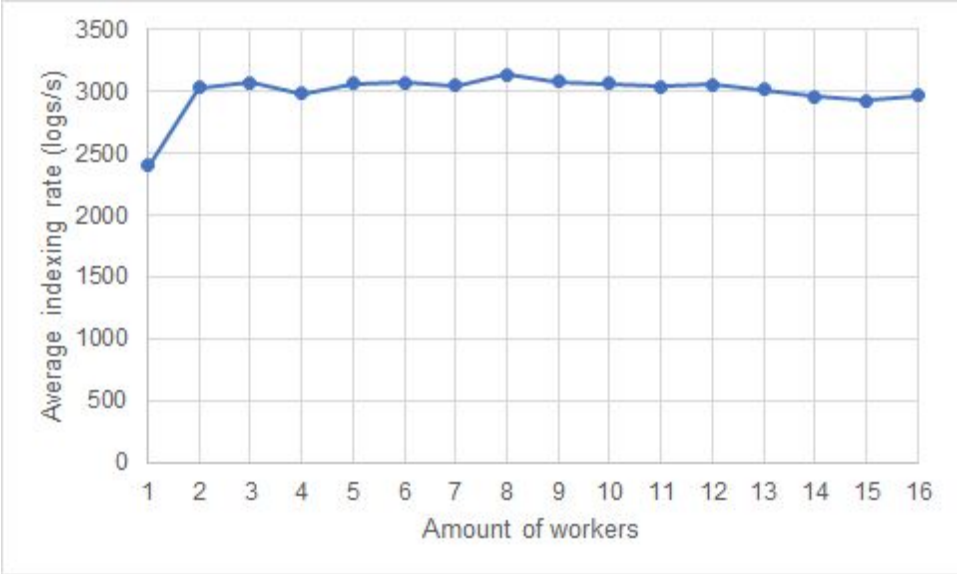

Fig. 11. Indexing rate (1250 entries per batch).

An analogous comparison can be done to special cases with pipeline workers. Here, the performance of just one worker is shown to follow almost a perfect logarithmical scale as the batch size increases, topping 2500 average indexed logs per second (see figure 12).

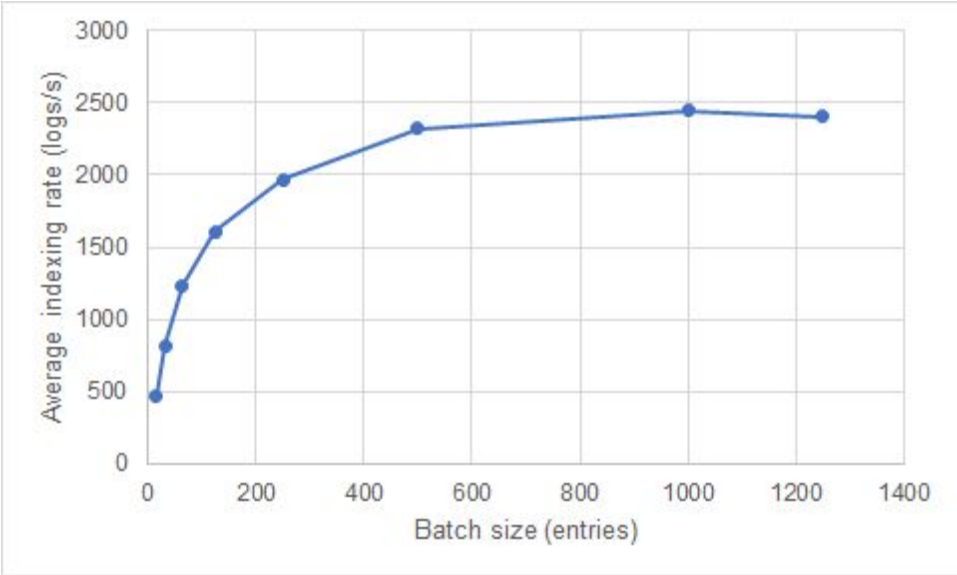

Fig. 12. Indexing rate (1 pipeline worker).

As more workers are available, this scale becomes steeper and the system limit is reached, as shown in figure 13.

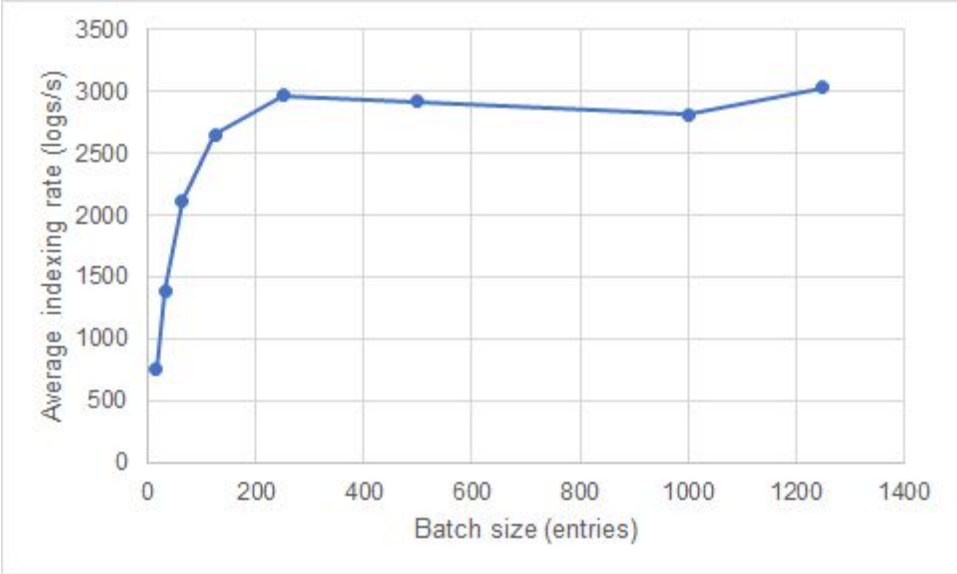

Fig. 13. Indexing rate (2 pipeline workers).

Finally, the graphs of 4 workers or more all show the same behavior as figure 14. That is, system limit reached with 3 or more workers.

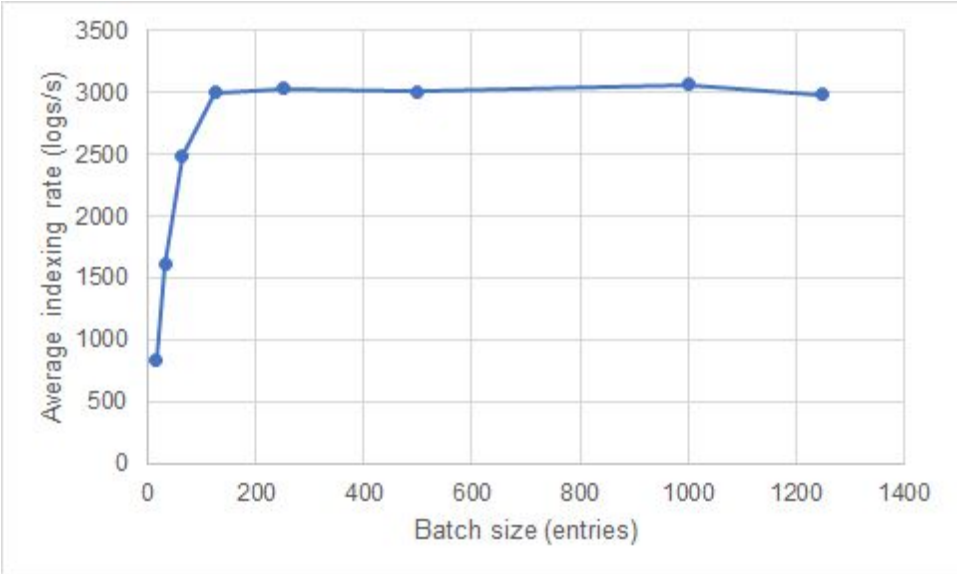

Fig. 14. Indexing rate (4 pipeline workers).

Conclusion

For this specific platform, the effect of the variation of the number of pipeline workers and the batch size on the indexing rate of the Elastic stack has been analyzed. As the maximum rate does not reach higher values than 3200 average indexes per second due to system limitations, a combination of 2 to 4 pipeline workers, and a batch size not greater than 500 entries, provides with the optimum resource usage in this scenario.

The flexibility of the test tools makes it possible to benchmark very heterogeneous configurations of the stack with minimal to no modifications. Thus, in future works, it is feasible to evaluate the effect of the introduction of a message queue in the architecture as a buffer between two Logstash instances, one working as a shipping instance, and the other as an indexing instance. It is also interesting to assess the behavior of Elasticsearch in such architecture, when there are more nodes available for horizontal scaling. In this case, more than one instance of the entries generator could be running to further stress test the system.

References

1. Innovation Accounting. (n.d.). Retrieved January 03, 2017, from https://leanstack.com/innovation-accounting/
2. Schnell, K., Puri, C., Mahler, P., & Dukatz, C. (2014). Teaching an old log new tricks with machine learning. *Big Data*, *2*(1), 7-11.
3. Product Overview. (n.d.). Retrieved January 03, 2017, from https://www.elastic.co/products
4. Elastic - Home. (n.d.). Retrieved January 03, 2017, from https://www.elastic.co/
5. Elasticsearch. (n.d.). Retrieved January 03, 2017, from https://www.elastic.co/products/elasticsearch
6. Logstash. (n.d.). Retrieved January 03, 2017, from https://www.elastic.co/products/logstash
7. Kibana. (n.d.). Retrieved January 03, 2017, from https://www.elastic.co/products/kibana
8. As A Service. (n.d.). Retrieved January 03, 2017, from https://www.elastic.co/cloud/as-a-service
9. Logstash Pipeline Architecture Discussion. (2016, July 21). Retrieved January 03, 2017, from https://www.youtube.com/watch?v=FPLHS9Pmgk0
10. Elasticsearch, Logstash, Kibana (ELK) Docker image. (n.d.). Retrieved January 03, 2017, from https://hub.docker.com/r/sebp/elk/
11. Devianthony. (2016, December 07). Deviantony/docker-elk. Retrieved January 03, 2017, from https://github.com/deviantony/docker-elk
12. Thies, S. (2015, April 28). 10 Elasticsearch metrics to watch. Retrieved January 03, 2017, from https://www.oreilly.com/ideas/10-elasticsearch-metrics-to-watch
13. Command-Line Flags | Logstash Reference [5.1] | Elastic. (n.d.). Retrieved January 03, 2017, from https://www.elastic.co/guide/en/logstash/current/command-line-flags.html