Performance Improvements of an Event Index Distributed System

Álvaro Fernández Casaní*[1], Juan Orduña[2], Santiago González de la Hoz[1]

[1] Instituto de Física Corpuscular (IFIC), Universidad de Valencia and CSIC, Spain, [2] Departamento de Informática, Universidad de Valencia , Spain

ABSTRACT

Modern scientific collaborations, like the ATLAS experiment at CERN, produce large amounts of data that need cataloging to meet multiple use cases and search criteria. Challenges arise in indexing and collecting billions of events, or particle collisions, from hundred of grid sites worldwide. In addition we face challenges in the organization of the data storage layer of the catalog, that should be capable of handling mixed OLTP (high-volume transaction processing updates) and OLAP (real-time analytical queries) use cases.

OBJECTIVES

In order to overcome the challenge on the distributed data collection of events, we have designed and implemented a distributed producer/consumer architecture, based on an Object Store as a shared storage, and with dynamic data selection. Producers run at hundreds of grid sites worldwide indexing millions of files summing up Petabytes of data, and store a small quantity of metadata per event in an ObjectStore. Then a reference to the data is sent to a supervisor, that signals consumers to retrieve the data at the desired granularity and consolidate at a central Hadoop based data backend.

In the area of the internal organization of the data, we propose an architecture based on a NoSQL backend storage, and new data schemas to better accommodate related data (reprocessings), avoiding duplicate information, and improving navigation. We propose applying memory caching techniques to improve access times for recent loaded data, which is usually the most accessed data by the end-user use cases.

INTRODUCTION

The ATLAS experiment at CERN [1] is producing, during its Run2 phase (2015-2018), in the order of 10^{10} events or particle collisions every year. This data is stored and distributedly reprocessed at different sites worldwide using grid technologies, to extract higher level information and store it in formats more suitable to different uses. A catalog of data (all events in all processing stages) is

therefore needed to meet use cases like (I) locate Individual events (event picking) depending on constraints, (II) make consistency checks, including detection of duplicates and overlaps, and (III) make analytic studies over large amounts of data. The EventIndex project [2, 3] is a metadata catalogue at event level which tries to exploit technologies such as Hadoop [6]. A small quantity of metadata per event is indexed, including identifiers (run/event numbers, trigger stream, luminosity block), the trigger pattern that made the event to be recorded, and references (pointers) to the events at each processing step in all permanent files on storage. We developed a producer/consumer architecture for the distributed data collection task of the EventIndex project with a messaging implementation [5] as depicted in figure 1, that has been indexing petabytes of input data, and has produced 150 TB of events meta data that are stored at the Hadoop infrastructure at CERN. This system, has efficiently handled more than 10^9 messages, but during high production campaings, we detected head of line blockings on the messaging broker. Since during the following runs starting in 2021 he production rates will be increased, we needed to explore other collection mechanisms.

MATERIALS AND METHODS – DISTRIBUTED DATA COLLECTION

During operation we detected head of line blockings on messaging brokers, as can be seen on figure 2 where messages accumulate in the backlog. There are periods when one Consumer is not getting any message even there is backlog at the broker, as can be seen in figure 3. In that cases even when there are new instances of the consumer launched against the brokers, the consumption rate does not increase. Messaging systems are designed to handle a large number of small messages, but our typical payload consists on large data files that have to be divided into smaller messages. This segmentation and re-assembly procedure is complex, and it has an effect on the brokers and consumers performance, and the scalability of the system.

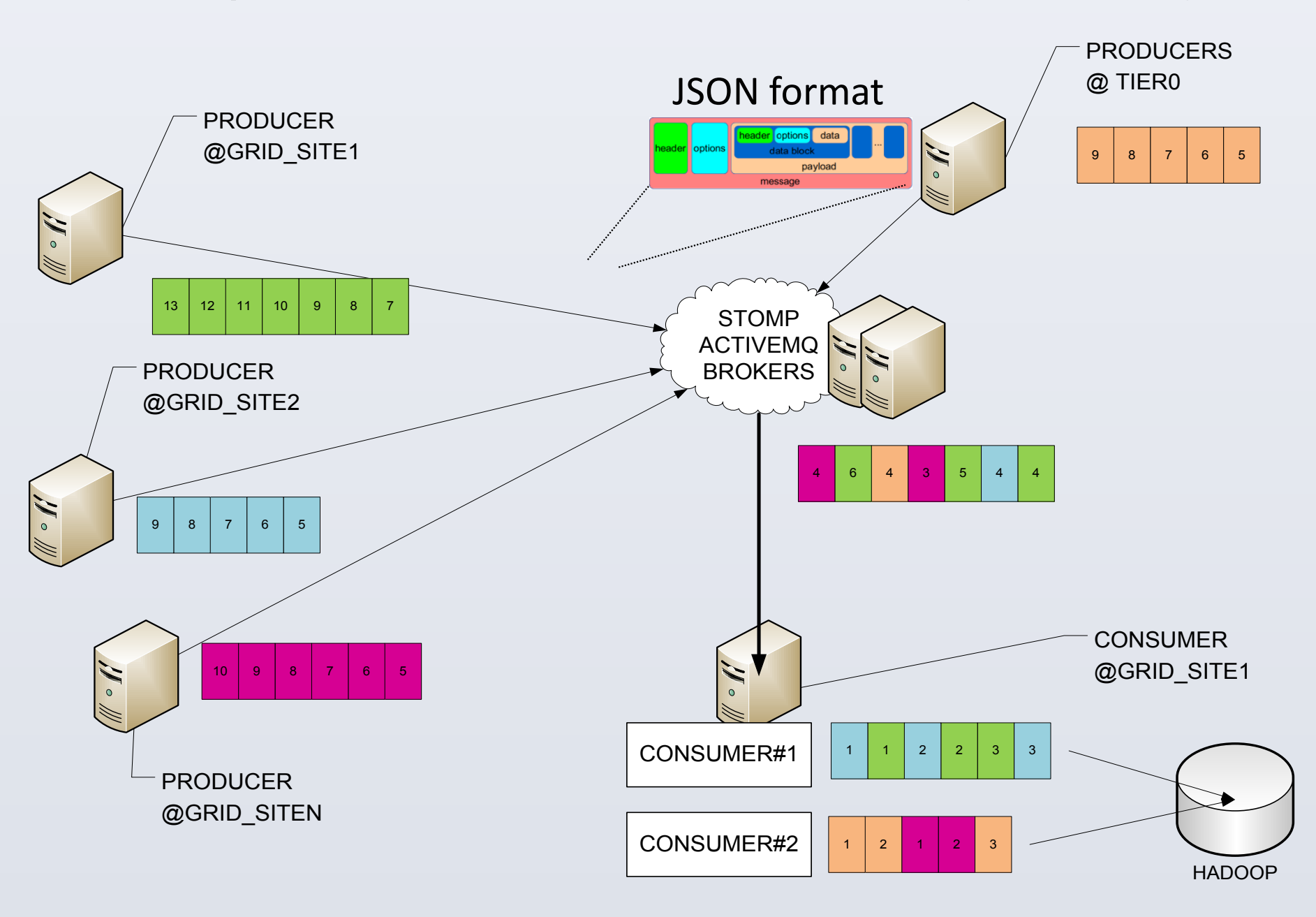

Figure 1. Messaging implementation of the data collection architecture

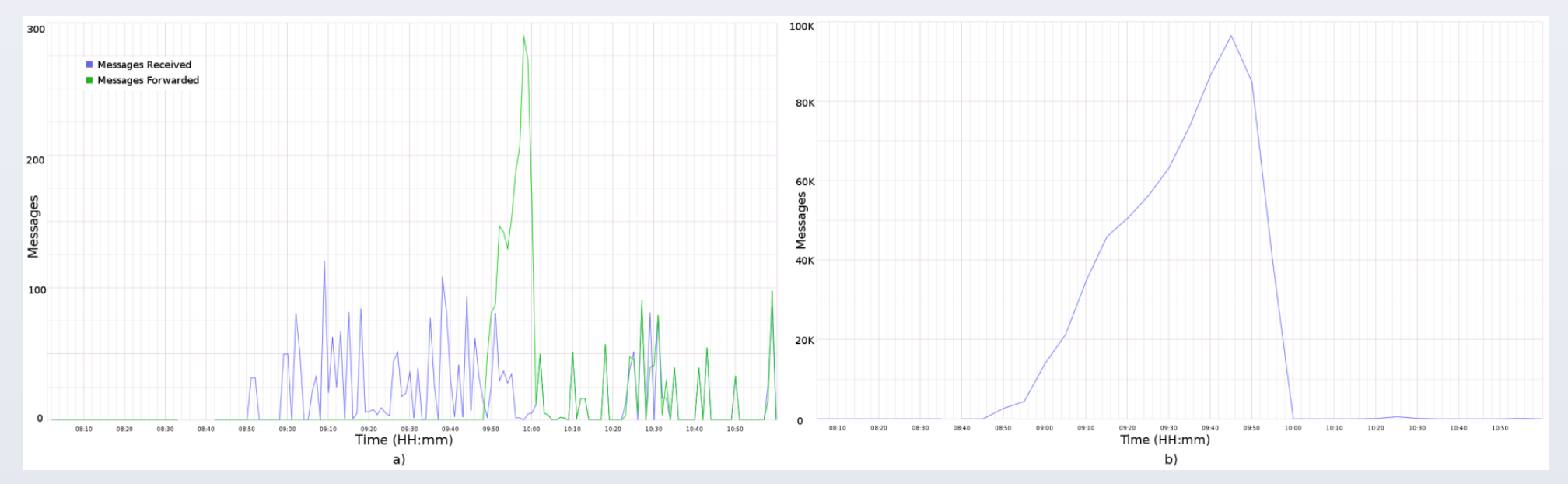

Figure 2: Head of line blocking detection on brokers a) Messages received (blue) and forwarded to consumers (green) by the brokers. b)Messages stored (not forwarded at a given time)

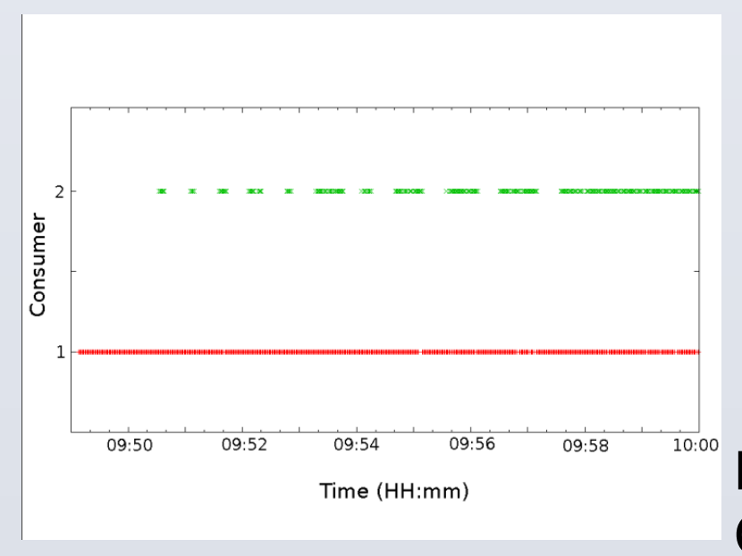

Figure 3: Messages consumed along time by Consumer 1 (red) and Consumer 2 (green)

In order to solve the complexity and scalability problems, we have proposed, designed and implemented a new distributed producer/ consumer architecture, based on Object Store (OBS) as a shared storage, and with dynamic data selection [4]. The producer now puts all the event index data in a OBS, without dividing the payload in various messages, and when it is done it sends a control and statistic message to a supervisor. The supervisor is in charge of selecting the valid produced information and signaling consumers to retrieve the appropriate data from the OBS system. This entity also takes into account the possibility of some fraction of the event processing not reaching its final state, as it was done with the validator in the Messaging scenario.

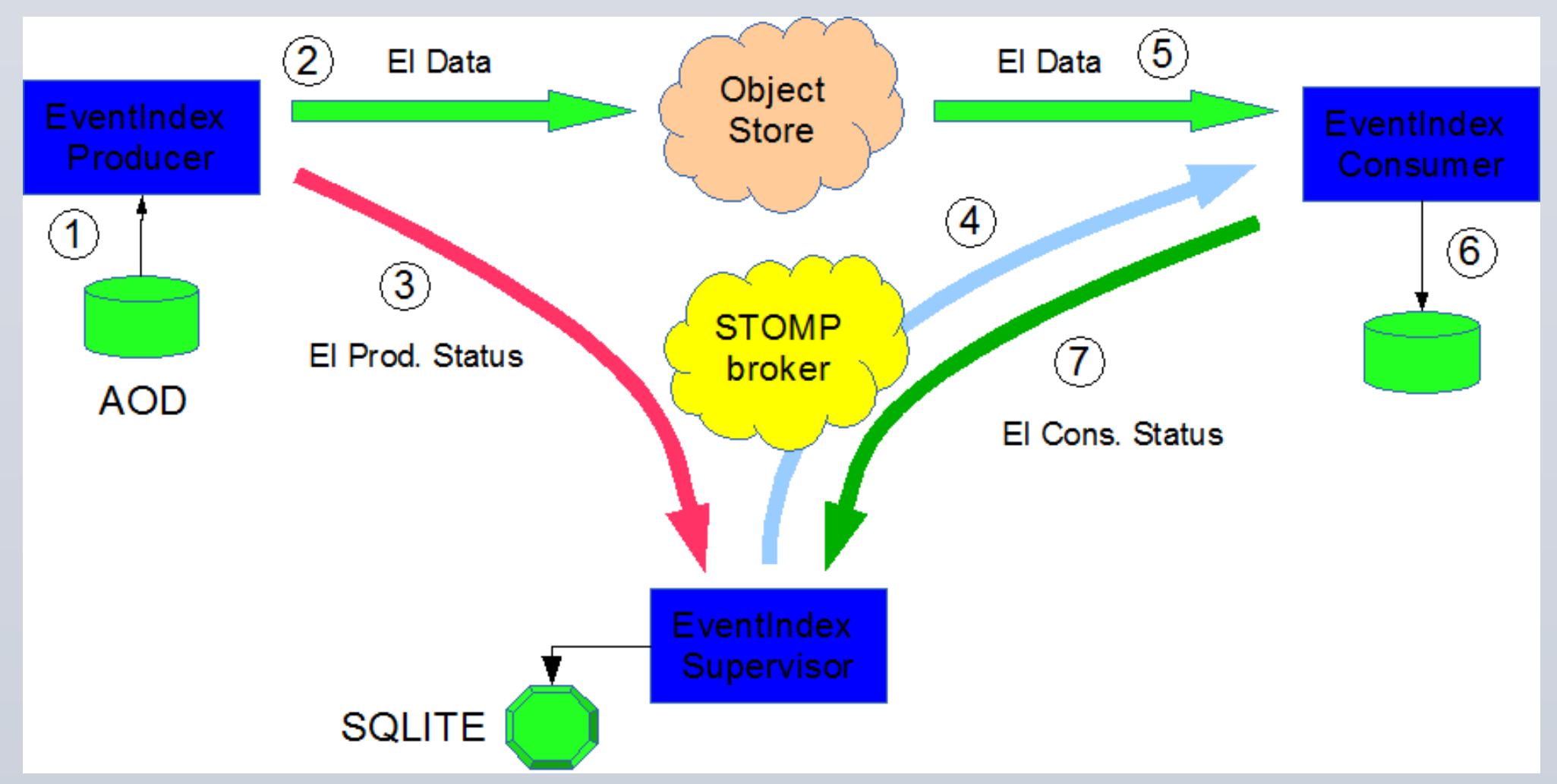

Figure 4. ObjectStore implementation of the data collection architecture

Now the difference is that this partial data is not being continuously pushed to the consumers. When reaching a desired processing granularity (for example indexing all the data from a dataset), the supervisor signals the consumer with a control message which contains all the info needed to retrieve the data by the latter. This allows a consumer to consolidate information in a single step, writing a unique file in HDFS filesystem, instead of having multiple files written by several consumers like in the messaging scenario. Only valid data is retrieved from the OBS and consolidated into bigger, more suitable files in the Hadoop HDFS filesystem. It reduces the amount of data that is consumed, and so the network usage from the OBS to the final HDFS backend. We can also avoid extra and expensive cleaning tasks on the Hadoop cluster.

RESULTS

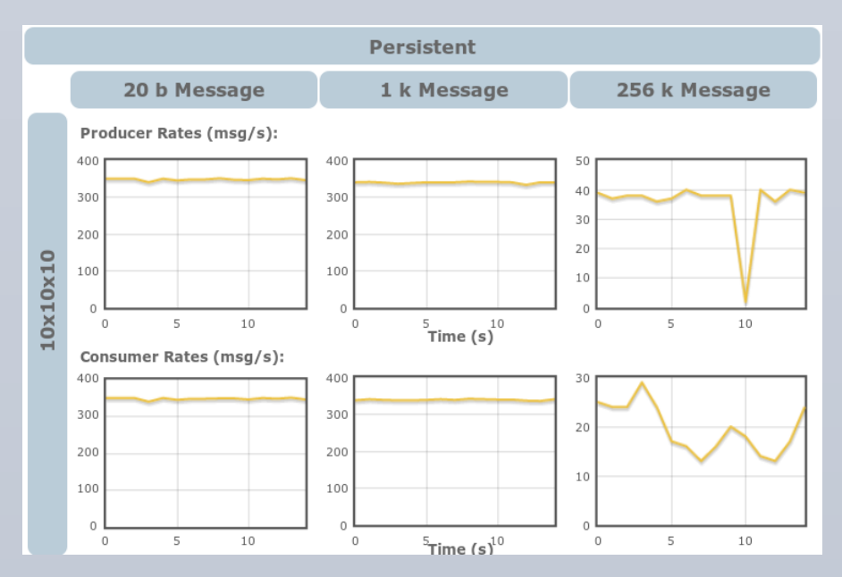

Figure 5. Messaging performance

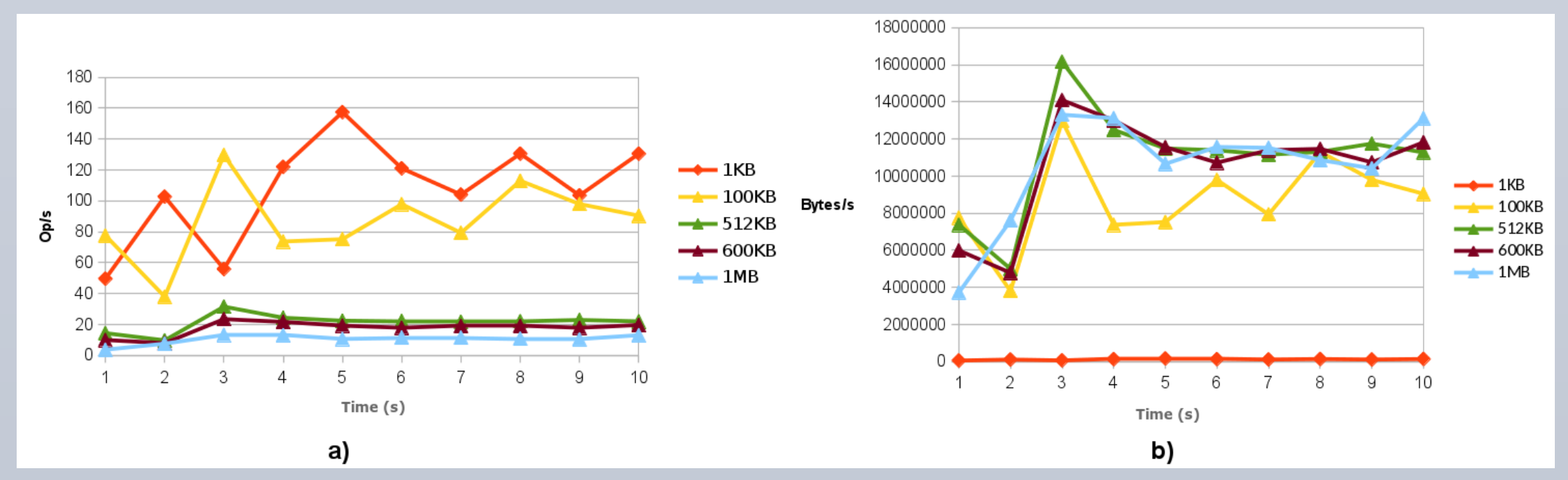

Figure 6. Object Store performance

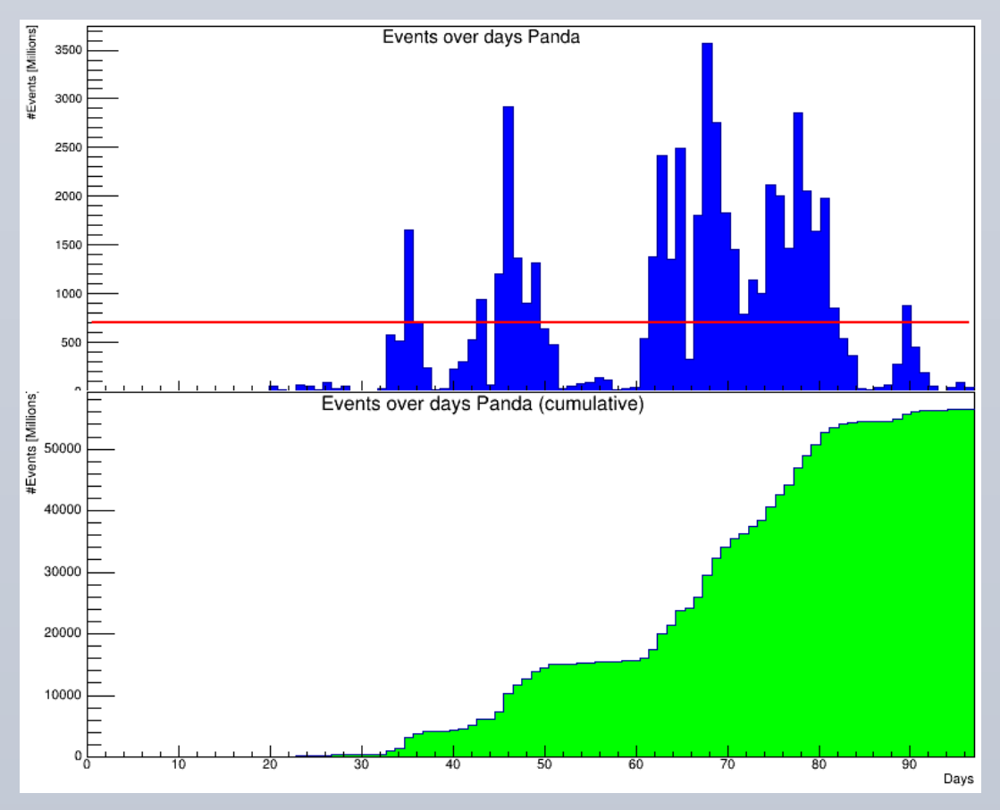

Figure 7.0 OBS system performance in collected events per day and totals.

Figure 6 shows the performance obtained with 10 simultaneous producers writing objects of different sizes, starting from 1KB up to 1MB. Figure 6 a) shows the system throughput. The throughput achieved is moderate, with a maximum of 150 operations per second obtained with 1KB objects. Figure 6 b) shows the bandwidth (measured in bytes per second written with different object sizes) in the Y-axis. Figure 6 b) shows that the achieved throughput is low with small 1KB objects, but it starts increasing with objects of 100KB, reaching a stable mean of about 12 MBytes/s with objects 512KB and up, with peaks up to 16MBytes/s.
Since the data used for benchmarking the OBS system come a real system in production, we can compare the performance of the messaging system shown in Figure 5 to the OBS approach in Figure 10 b). From this comparison, we can state that for small payload sizes the messaging system yields a better throughput than the OBS system. However, Figure 10 b) shows that for payload sizes equal or greater than 100KB the OBS system yields a better bandwidth than the messaging system, which is true for our typical producer payload in the order of MB.
In Figure 7 we can see the number of collected events with the new OBS system during 3 months in production, summing up a total of 60 Billion events indexed, with peaks of 3.5 Billion events per day. The new OBS consumer improved performance an order of magnitude compared with the Messaging one, being now capable to absorb these peaks online. The ObjectStore based solution is now the reference implementation for the ATLAS EventIndex project, and is being currently used in production since 2018.

CONCLUSIONS

We have presented a new pull-model approach for the distributed data collection of the ATLAS EventIndex project, based on an Object Store as a shared storage and with dynamic data selection. It must be noted two key differences in this new approach. First, the entire payload from a given producer can be potentially stored within a single object regardless of its size, avoiding complex issues like message groups and transactions. This avoids blockings due to this matter, and so better workload distribution and scalability adding new consumers when necessary. This was not effectively possible with the messaging approach and in addition this allows us to use different data encodings and compression, reducing the amount of conveyed data. Second, the behavioural change from a push-model to a pull-model. This model allows to use the OBS as a temporary storage, eliminating the need to consume duplicated produced data. The reduction of complexity and the resource usage, and better performance of the distributed data collection, has improved the experience for final users, that have seen reduced the Traversal time, or latency, of the datasets indexed data. Overall the results show that the new approach can efficiently support large-scale data collection for big data environments, like the next runs of the ATLAS experiment at CERN.

REMAINING OBJECTIVES

The second objective of this thesis is to design and implement a data storage layer exploring these systems and capable to handle OLTP and OLAP mixed workloads, to satisfy the described use cases, and which improve the usability and performance. We are aiming to avoid duplicate information, and provide a unique and coherent dataset for all use cases and workloads

REFERENCES

[1] ATLAS Collaboration. 2008. The ATLAS Experiment at the CERN Large Hadron Collider. Journal of Instrumentation 3, 08 (2008), S08003.

[2] D. Barberis, S.E. Cárdenas Zárate, J. Cranshaw, A. Favareto, A. Fernández Casaní, E.J. Gallas, C. Glasman, S. González De La Hoz, J. Hřivnáč, D. Malon, F. Prokoshin, J. Salt Cairols, J. Sánchez, R. Többicke, and R. Yuan. 2015. The ATLAS EventIndex: Architecture, design choices, deployment and first operation experience. Journal of Physics: Conference Series 664, 4 (2015), 042003.

[3] D. Barberis, J. Cranshaw, A. Favareto, A. Fernández Casaní, E. Gallas, S. González de la Hoz, J. Hřivnáč, D. Malon, M. Nowak, F. Prokoshin, J. Salt, J. Sánchez Martínez, R. Többicke, and R. Yuan. 2016. The ATLAS EventIndex: Full chain deployment and first operation. Nuclear and Particle Physics Proceedings 273-275 (2016), 913–918.

[4] A Fernandez Casani, D Barberis, A Favareto, C Garcia Montoro, S Gonzalez de la Hoz, J Hřivnáč, F Prokoshin, J Salt, J Sanchez, Többicke, R Yuan, and ATLAS Collaboration. 2017. ATLAS EventIndex general dataflow and monitoring infrastructure. Journal of Physics: Conference Series 898, 6 (2017), 062010.

http://stacks.iop.org/1742-6596/898/i=6/a=062010

[5] J Sánchez, A Fernández Casaní, and S González de la Hoz. 2015. Distributed Data Collection for the ATLAS EventIndex. Journal of Physics: Conference Series 664, 4 (2015), 042046.

[6] Tom White. 2012. Hadoop: The Definitive Guide. O'Reilly Media, Inc.

ACKNOWLEDGEMENTS

This work has been supported by Spanish MINECO and EU ERDF programs under grants FPA2013-47424-C3,01,02,03, and TIN2015-66972-C5-5-R.

(*)Corresponding author email: alvaro.fernandez@ific.uv.es